mi computer

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

TOMO 5

Editorial Delta, S.A.


mi COMPUTER

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, S. Tarditti, F. Martín
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

ISBN: 84-7598-007-4 (tomo 5) 84-85822-82-X (obra completa)
Depósito legal: B. 52/1984
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona)
Impreso en España - Printed in Spain - 1984

mi COMPUTER

VOLUMEN 5

Editorial Delta, S.A.

Creación de melodías

Éste es el primer capítulo de una serie en la que analizaremos el MIDI (interface digital para instrumentos musicales)

La música y las ciencias exactas guardan una relación que se conoce desde hace muchos siglos. El matemático griego Pitágoras pesó un grupo de martillos de herrero para averiguar por qué parecían sonar de forma "melodiosa" al golpear contra el yunque. Descubrió que un martillo de la mitad de peso que otro producía un sonido de exactamente el doble de frecuencia, o una octava más alto. Con ello se estableció el primer principio que rige las relaciones de frecuencia en música.

En la Edad Media los compositores llenaban las catedrales con el sonido de misas y motetes (composiciones corales polifónicas) que estaban proporcionados rítmica y numéricamente con el mismo nivel de precisión que la arquitectura de las propias catedrales. A menudo su música era tan compleja que se creía que sólo los oídos de Dios eran capaces de apreciar las relaciones numéricas, mientras los meros humanos escuchaban música. Y cualquiera que haya presenciado una función de música en vivo (de casi cualquier tipo) habrá observado que los músicos cuentan "1,2,3,4; 2,2,3,4", en un susurro, antes de comenzar a tocar.

Por tanto, fue muy natural que los mundos de la informática y la música se superpusieran. En la actualidad, un desarrollo que está causando gran expectación es el MIDI (*Musical Instrument Digital Interface*: interface digital para instrumentos musicales). Esta unidad está diseñada para permitir que cualquier sistema digital, incluyendo microordenadores, controle las funciones de otro. Como la mayoría de los instrumentos musicales electrónicos que se están fabricando en la actualidad son digitales, se abre a los usuarios de un micro personal todo un nuevo campo de posibilidades.

Pero el MIDI no es una caja mágica. No convierte de la noche a la mañana al usuario de un micro en un Vangelis o un Stevie Wonder. Los conocimientos musicales y la imaginación siempre producen los mejores resultados, tanto si la música se ejecuta en un banco de sintetizadores conectados en interface como en una guitarra acústica.

Para comprender la clase de instrumentos musicales para que el MIDI está pensado y cómo se produce la música electrónica, hemos de remontarnos medio siglo atrás. Ya antes de la segunda guerra mundial los músicos habían empezando a experimentar con *generadores sine-tono* simples. Éstos eran dispositivos eléctricos que hacían vibrar una tira metálica, produciendo de ese modo un tono constante que podía variar en altura. Este sonido se solía utilizar para la ambientación musical de las películas de ciencia-ficción en la década de los cincuenta, para sugerir una atmósfera misteriosa o futurista. Aún hoy se escucha en los altavoces de los televisores como una señal para que los telespectadores apaguen los aparatos cuando acaba la emisión. Los primeros órganos Hammond que se comercializaron en los años treinta eran electrónicos y utilizaban esta clase de sonido.

Marcus Wilson-Smith

Música rupturista
La nueva música exige una nueva notación. Las partituras de Stockhausen, con sus representaciones pictóricas de los sonidos y las directrices gráficas de sincronización, no tienen ninguna relación con las partituras clásicas y, en realidad, fueron pensadas para que se parecieran a los diagramas de circuitos eléctricos

Pero fue el *boom* de la electrónica que se produjo durante la segunda guerra mundial, específicamente el desarrollo de la grabadora de cinta por parte de los alemanes, lo que posibilitó que los músicos crearan y manipularan el sonido de forma bastante diferente. Esto se podía realizar empalmando cinta magnetofónica analógica en la cual ya se había grabado sonido, ya fuera "musical" o de cualquier otro tipo. Estos diminutos retazos de cinta se combinaban entonces esmeradamente para producir un collage de eventos sonoros. Esta "nueva música" representó una ruptura con todas las reglas escritas acerca de teoría musical convencional. Fascinaba a algunos oyentes en la misma medida en que desagradaba a otros.

"Doctor" pionero

El tema más clásico de la música electrónica es, sin ninguna duda, *"Doctor Who"*, escrito para el BBC Radiophonic Workshop en 1962 por Ron Grainer (al que vemos a la izquierda de la fotografía, bromeando con algunos de sus compañeros de plató en la serie *Maigret* de la BBC Television)

Al mismo tiempo se fueron difundiendo y haciendo más controlables dispositivos para alterar y distorsionar los tonos del oscilador, originalmente sencillos, y para filtrar y modular el resultado. Durante la década de los cincuenta, compositores como Stockhausen, en Alemania, trabajaban afanosamente en pequeños estudios junto a las estaciones de radio locales, produciendo música electrónica "pura". En París, investigando en estrecha relación con ingenieros de sonido de la ORTF (la empresa de radiotelevisión de Francia), Pierre Schaeffer se convirtió en un pionero de lo que él denominó *musique concrète*, música de collage que utilizaba sonidos cotidianos del mundo real.

En Estados Unidos, Bell Telephone Laboratories construyó lo que probablemente haya sido el primer sintetizador. Ocupaba varias habitaciones y su objetivo fundamental era el estudio de la síntesis de la voz humana. La empresa sabía que sus operadores telefónicos, de distintos lugares del país, con frecuencia no comprendían bien los acentos de unos y otros, lo que daba lugar a una elevada proporción de conexiones falsas y números equivocados. Ellos creían, quizá con cierto exceso de optimismo para aquellos tiempos, que una voz sintetizada aceptada a escala universal acabaría con el problema. Numerosos músicos norteamericanos de la actualidad recibieron su formación básica en electrónica en aquel entonces, por cortesía de la empresa Bell.

En Gran Bretaña se estaban realizando trabajos similares aunque a una escala menos ambiciosa. No obstante, el BBC Radiophonic Workshop (taller radiofónico de la BBC) produjo, a principios de los años sesenta, uno de los más grandes clásicos de la música electrónica de todos los tiempos: la banda sonora para la serie de televisión *Doctor Who*.

La primera incursión en el campo de la música por ordenador se produjo ya en 1957, cuando Lejaren Hiller introdujo un conjunto de instrucciones en el ordenador Illiac de la Universidad de Illinois. Estas instrucciones se transformaron en cuatro grupos de datos técnicos que se transcribieron entonces a notación musical. El resultado fue una obra en cuatro movimientos para cuarteto de cuerdas llamada *Illiac suite*. La música en sí, aunque bien adaptada para su ejecución con cello, viola y dos violines, sonaba vaga y como sin rumbo. Sin embargo, no es difícil encontrar otras piezas musicales, producidas de forma convencional por compositores de la época, que suenan aún mucho peor.

Pocos años después Hiller creó otra obra, esta vez utilizando el ordenador IBM 7090. Diseñó un esquema de programación denominado MUSICOMP (*MUsic SImulator-Interpreter for COMpositional Procedures*: intérprete-simulador musical para procedimientos de composición), que permitió una mayor flexibilidad y variedad en el trabajo para llegar a la composición final. A la misma la bautizó *Computer cantata (Cantata para ordenador)* y está escrita para un vocalista con sonidos electrónicos grabados. Nuevamente, la música resulta interesante a ratos en lugar de ser subyugante. Pero Hiller había demostrado a sus colegas que el ordenador se podía utilizar de forma creativa.

Su trabajo representó sólo una parte de una intensa investigación que se llevó a cabo en las universidades norteamericanas en los años siguientes. John Chowning, otro pionero, empleó más tarde un ordenador para explorar cómo se percibe el sonido mientras la fuente que lo produce se desplaza de un lugar a otro. La utilización de sus trabajos de investigación por parte de Yamaha ha tenido una incidencia directa en el tipo de sintetizador que se está produciendo en la actualidad.

Con la excepción de la música para crear ambientes de ciencia-ficción, la música electrónica permaneció en el ámbito de la música clásica durante varios años y el público tomó conciencia de este cambio de enfoque y de técnica a través de los compositores de vanguardia. En los típicos conciertos de nueva música de los años sesenta participaban varios intérpretes, algunos de ellos tocando instrumentos convencionales, otros dedicados a procesar el sonido de aquellos instrumentos con unidades de separación de frecuencia y filtros. Todos los músicos, incluyendo a los "técnicos", seguían una partitura, pero ésta guardaba poco parecido con la notación musical estándar.

Además de las directrices novedosas, como las que describían las posiciones de los micrófonos y las variaciones de los filtros, los compositores intentaban obtener indicaciones visuales del aspecto que tenían estos nuevos sonidos al ejecutarlos. En algunos casos los músicos tocaban utilizando partituras

Espíritu precursor

Probablemente más conocido por su trabajo con Roxy Music a principios de los años setenta, Brian Eno fue un pionero en la utilización de los primeros sintetizadores. Después de dejar el grupo, en 1973, Eno ha sido uno de los puntales de la música electrónica de vanguardia y música *mood*. También ha colaborado con figuras del calibre de David Bowie y Robert Fripp. Más recientemente, Eno ha trabajado en partituras para televisión y cine y, junto con su hermano, ha desarrollado una partitura para la película de archivo de la NASA sobre aterrizaje lunar

que más bien parecían hojas garabateadas por un diseñador gráfico. Este problema (el de cómo explicar la ejecución de la música, qué lenguaje emplear y cómo visualizar el resultado) persiste en los sistemas de música digital en los años ochenta.

A medida que fue transcurriendo la década de los sesenta, los músicos pop de la floreciente cultura juvenil empezaron a pasarse más tiempo en los estudios de grabación, y también comenzaron a experimentar con música electrónica. El ejemplo clásico es el de los Beatles, quienes hallaron en George Martin no sólo un experto ingeniero de grabación, sino también a un músico que había estado profesionalmente atento a los desarrollos que se habían producido en el campo de la música clásica. Él alentó a los Beatles para que utilizaran todo el estudio como un instrumento musical y al poco tiempo ya estaban aplicando técnicas de collage con cintas e incorporando sonido sintetizado.

Algunos músicos ganaron su prestigio mediante la utilización de determinadas unidades para proceso de sonido. Un guitarrista como Jimmy Page modeló su estilo en base al de los músicos negros norteamericanos de los años cuarenta, pero utilizando una serie de controles de distorsión produjo un sonido de identificación inmediata como el de Led Zeppelin. Esto, junto al empleo por Jimi Hendrix de mecanismos de autoalimentación de sonido y filtros de barrido rápido (conocidos hoy como "pedales wah-wah"), constituyó la base del *heavy metal*.

En los años setenta las distintas unidades para generación y proceso del sonido, que habían estado disponibles desde los años cincuenta, comenzaron a incorporar diseños transistorizados. Se volvieron más pequeñas, más portátiles y, como resultado, menos restringidas al ámbito de los estudios. Los guitarristas podían tocar en vivo empleando un conjunto de pedales para efectos especiales. Al poco tiempo, los organistas y pianistas tuvieron acceso a sintetizadores de precio asequible que podían utilizar en los escenarios.

Típicamente, estos sintetizadores incluían un juego de osciladores regulables, moldeadores de envolventes (para crear las características del sonido: ataque, sostenimiento y decaimiento), filtros variables, moduladores que podían separar las señales en frecuencias nuevas y generadores de ruido. Así como Jimi Hendrix había sido un modelo para los guitarristas, Brian Eno se convirtió en el modelo para los músicos de sintetizador, principalmente debido a sus vinculaciones con la "nueva música" de la vanguardia clásica.

Al mismo tiempo, los equipos de los estudios de grabación se volvieron más sofisticados, puesto que los músicos buscaban que el estudio les proporcionara algo del proceso de producción que ellos no pudieran crear en el escenario. La mesa de mezclas, ahora diseñada para canalizar grabaciones sucesivas en 16 o 24 pistas de cinta, era todavía demasiado grande como para llevarla de un lado a otro, y la instalación de muchas de las unidades de proceso llevaba su tiempo. En Estados Unidos y en Gran Bretaña surgió una nueva generación de productores. Por lo general habían empezado como ingenieros y estaban mucho más familiarizados con el equipo que los músicos que les habían pagado para que les proporcionaran "el sonido correcto".

En Jamaica los ingenieros empezaron a utilizar la mesa de mezclas como si fuera un instrumento. Las canciones acabadas, grabadas en cinta de pistas múltiples, se volvían a desmontar en sus pistas rítmicas individuales. Las contribuciones originales, vocales o instrumentales, se utilizaban entonces como materia prima para usarlas en la mezcla o eliminarlas de la misma en un estilo que dependía mayormente de la reverberación y las unidades de retardo de señal; éste fue el estilo *dub*.

El advenimiento de los sintetizadores digitales supuso la posibilidad de codificar sonidos no electrónicos. Este proceso se conoce como "muestreo" (*sampling*). Baterías eléctricas como el Linn se convirtieron en objetos muy buscados en los estudios y enseguida se incorporaron a las actuaciones en vivo. A mediados de los ochenta, el muestreo y la manipulación del sonido se han convertido en la "naturaleza del arte", y los estudios y escenarios bien equipados por lo general disponen de más aparatos de equipos digitales que de instrumentos analógicos. Grupos de gran éxito, como el británico Culture Club, combinan sus propias aptitudes musicales para la escritura de canciones con las técnicas digitales de productores como Steve Levine, quien utiliza instrumentos y unidades de proceso que valen decenas de miles de libras.

La necesidad de una interface que conectara a un instrumento con otro, o que ampliara la capacidad de un sintetizador mediante la adición del sistema operativo y la memoria de un microordenador, determinó la unión de los fabricantes de instrumentos musicales. Éstos produjeron las primeras especificaciones del MIDI en abril de 1983 y, desde entonces, pocas empresas se han atrevido a anunciar un nuevo sintetizador que no sea compatible con el MIDI. En el próximo capítulo estudiaremos con detalle los antecedentes y el desarrollo del MIDI.

Productor de ases

Si bien es conocido como productor del grupo británico Culture Club, Steve Levine es más famoso por su habilidad para combinar música electrónica y voces humanas para producir pop *seamless* (bien engranado). Levine fue uno de los primeros productores de Gran Bretaña que hicieron uso del tambor Linn, un sintetizador digital programable, y recientemente ha desarrollado nuevas técnicas de grabación digitales. Levine ha hecho una gran utilización de sintetizadores y otros equipos de música digitales en su reciente single, *Believin'*, que escribió conjuntamente con Boy George, integrante de Culture Club

Errores típicos

La detección y corrección de errores es un aspecto importante del diseño de programas: veamos cómo tratarlos

Existen muchas fuentes de error potenciales en cada una de las etapas de la creación de un programa, desde su especificación, pasando por el diseño y la codificación, hasta la comprobación. Con frecuencia se introducen errores en las etapas de especificación y diseño si se presta poca atención a la naturaleza del problema y si no se tiene el suficiente cuidado para asegurar que el programa haga exactamente lo que se supone debe hacer. Podemos reducir las posibilidades de que se produzcan estos errores siguiendo los métodos de diseño estructurado que esbozamos previamente en el curso (véase p. 956). Es probable que surjan otros errores cuando se traduce el diseño a código —una digitación deficiente puede crear errores (*bugs*), ¡como sabrá cualquiera que alguna vez haya escrito mal el nombre de una variable!— y hasta la comprobación y la depuración pueden dar lugar a otros problemas cuando la corrección de un fallo origina otros.

Pero es en las interfaces (entre rutinas y entre el programa y su usuario) donde se producen la mayoría de los errores. Se debe tener especial cuidado en asegurar que todos los valores que se pasen por estas interfaces sean del tipo de datos correcto y estén comprendidos en el ámbito de valores requerido por el programa. Los valores se pueden verificar dentro de la misma rutina que los pasa o bien en la rutina que los acepta; este proceso de verificar los valores a medida que pasan entre rutinas se conoce por *cortafuegos*.

Para asegurar que los valores que salen de una rutina estén dentro de la escala apropiada y sean del tipo de datos correcto, se debe comprobar si la salida depende de un valor introducido por un usuario o leído de un archivo. Los valores que se introducen en una rutina siempre se deben comprobar. Las subrutinas se pueden diseñar para dar un conjunto de salidas bien definidas, pero los seres humanos no funcionan tan automáticamente y tienden a dar una amplia gama de respuestas diferentes ante cualquier pregunta dada, de modo que en todas las rutinas que acepten datos de operadores humanos se deben colocar comprobaciones rigurosas. De forma similar, los archivos de datos se pueden corromper o leer equivocadamente, de modo que se deben colocar comprobaciones en todas las rutinas de manipulación de archivos.

No es frecuente que los errores hagan que el programa se rompa. Cuando lo hacen es porque el programa ha quebrantado alguna regla de lenguaje (utilizando ilegalmente un operador, como, p.ej., en RESULTADO=PRIMERO$+SEGUNDO$) o una regla del sistema operativo (abrir demasiados archivos a un tiempo, p. ej.). El código que sigue parece que es un programa perfectamente legítimo:

```
10 FOR CONTADOR=1 TO 10
20       SUMA=SUMA+1
30       PRINT CONTADOR,SUMA
40       GOTO 10
50 NEXT CONTADOR
```

Sin embargo, es un algoritmo que no acaba nunca y romperá el programa debido a la forma en que funciona el lenguaje. En este caso, éste (BASIC) utiliza la "pila" para llevar el registro de los bucles FOR...NEXT, sumando a la pila cada vez que se pasa por el principio de bucle (línea 10). En este programa, a la línea 50 (con la instrucción NEXT que disminuiría la pila) no se llega nunca, y, por tanto, la pila se va llenando poco a poco hasta que finalmente se genera un mensaje *stack overflow* (desbordamiento de capacidad de la pila) y el intérprete interrumpe el programa. Por lo general, los errores de este tipo son fáciles de detectar, pero si aparecieran en secciones de código de poco uso sería necesaria una comprobación exhaustiva para descubrirlos.

Un tipo de error más molesto es el que permite que un programa se ejecute normalmente pero dando resultados erróneos. A modo de ejemplo

Lista de comprobación de errores

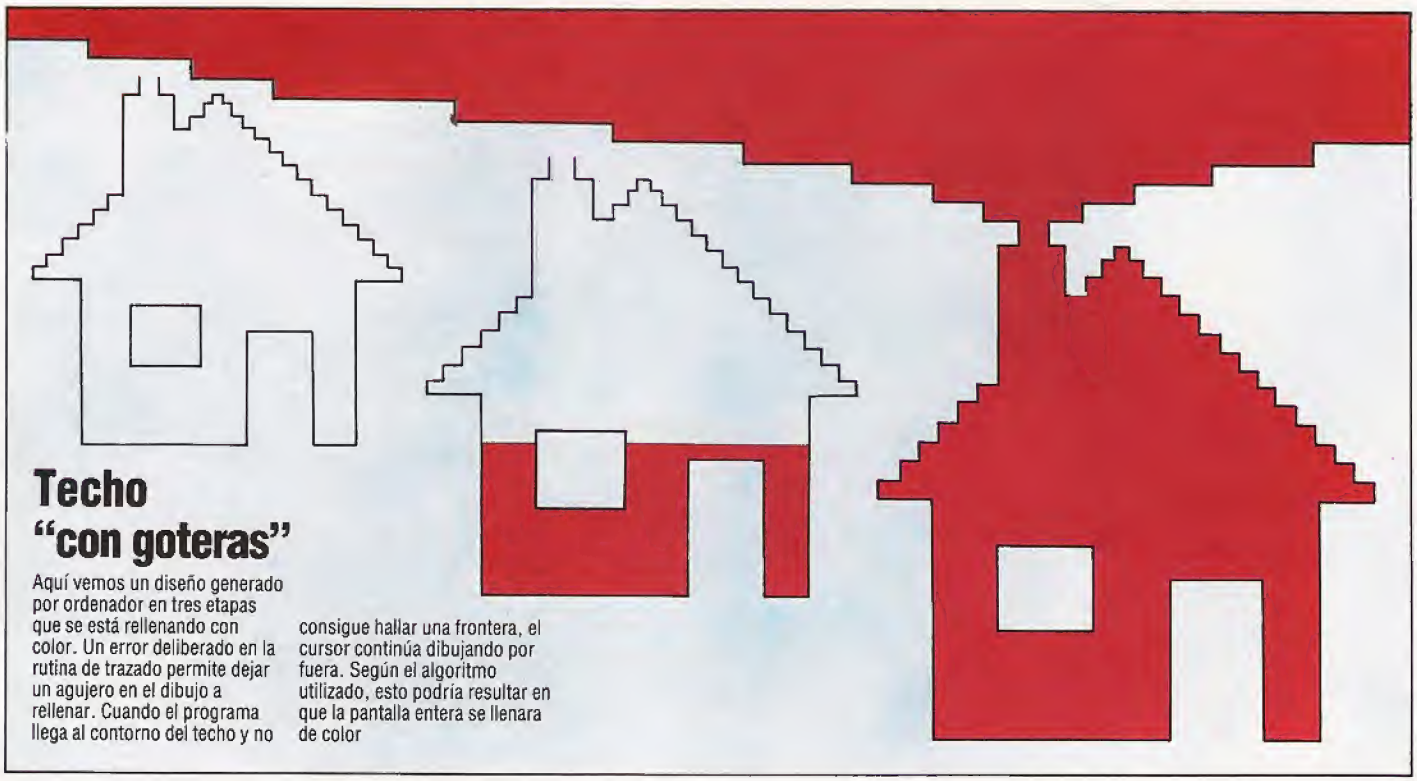
Un enfoque estructurado lógico es lo fundamental para evitar errores y acortar el proceso de depuración; la siguiente lista de comprobación de errores (elaborada a partir de una idea de G.J. Myers en *The art of software testing*) es un ejemplo abreviado de un enfoque de este tipo

Variables
1 ¿Son exclusivos todos los nombres de las variables, teniendo en cuenta que muchos intérpretes utilizan solamente los dos primeros caracteres de cualquier nombre?
2 ¿Se ha vuelto a emplear alguna variable (especialmente los contadores de bucles o los parámetros de subrutinas) mientras su contenido aún era significativo?
3 ¿Están los subíndices de las matrices dentro de los límites, y son números enteros?
4 ¿Empiezan los subíndices de las matrices en el elemento cero o en el elemento uno?

Cálculos
1 ¿Producen los cálculos resultados numéricos o alfanuméricos? Y los resultados, ¿se les asignan a variables numéricas o alfanuméricas?
2 ¿Algún cálculo da como resultado un número demasiado pequeño o grande para que el ordenador lo manipule? ¿Puede esto producir un error de "división por cero"?
3 ¿Pueden ser significativos los errores de redondeo?
4 ¿Están todas las operaciones en una expresión que se ejecute por el orden lógico correcto, en contraposición al impuesto por la precedencia de operadores aritméticos?

Comparaciones
1 ¿Las matrices se comparan siempre con series, y los números con números?
2 ¿Importa que una variable alfanumérica o en serie de verificación esté total o parcialmente en mayúsculas o minúsculas?
3 ¿Se están comparando series de longitud desigual? Y la diferencia en longitud, ¿importa más o menos que las diferencias en caracteres?
4 ¿Se están mezclando adecuadamente operadores booleanos y de comparación? A > B OR C, por ejemplo, no es lo mismo que A > B OR A > C.
5 La precedencia de operadores booleanos y de comparación, ¿afecta a la ejecución de cualquier expresión de comparación?

Control
1 Los bucles y algoritmos, ¿terminan sea cual sea el estado de las variables?
2 ¿Tienen los bucles y las rutinas un único punto de entrada y salida cada uno?
3 Cuando fracasa una sentencia IF...THEN, ¿el control pasa a la siguiente sentencia del programa o a la siguiente línea del programa?
4 ¿Qué sucede si no se satisface ninguna de las condiciones de una sentencia de bifurcación múltiple?



Techo "con goteras"

Aquí vemos un diseño generado por ordenador en tres etapas que se está rellenando con color. Un error deliberado en la rutina de trazado permite dejar un agujero en el dibujo a rellenar. Cuando el programa llega al contorno del techo y no consigue hallar una frontera, el cursor continúa dibujando por fuera. Según el algoritmo utilizado, esto podría resultar en que la pantalla entera se llenara de color

Kevin Jones

hemos elegido un patrón de relleno que traza un dibujo en la pantalla y después lo rellena con color. Las rutinas de relleno buscan las fronteras del dibujo. Cuando se alcanza una frontera, el ordenador hace dar la vuelta al cursor y continúa dibujando hasta alcanzar otra frontera. Para que una rutina de relleno funcione, las fronteras deben estar bien definidas y completas. Dicho en otras palabras, en el esquema del dibujo no puede haber espacios abiertos ya que, de lo contrario, la rutina de relleno desparramaría el color por fuera de las fronteras.

Las versiones de BASIC que utilizan la mayoría de los micros personales hacen que el tratamiento de errores resulte relativamente sencillo, produciendo mensajes de error claros y concisos y permitiendo que un programa roto continúe después de haber modificado los valores de las variables por el teclado, facilidad muy útil cuando se está depurando un programa. La mayoría de las versiones del BASIC permitirán el empleo de una instrucción como ON ERROR GOTO para transferir el flujo de control a una rutina especial para tratamiento de errores y controlar así errores que de otra forma serían decisivos. Se realiza incluyendo una línea como ésta:

```
30 ON ERROR GOTO 20000: REM rutinas para tratamiento de errores
```

cerca del comienzo del programa. Cualquier error hará entonces que el programa actúe como si se hubiera encontrado con la instrucción GOTO 20000. Por lo general ON ERROR también modifica dos variables; la primera de ellas almacena un código de error que indica la clase de error que se ha producido, y la otra simplemente retiene el número de línea en la cual se ha producido el error. Los nombres otorgados a estas variables y los códigos de error resultantes variarán de una máquina a otra, de modo que debe consultarse el manual. Al producirse un error, el flujo del programa se desvía hacia la línea 20000, ahí se identifica el error a partir del número hallado en la variable correspondiente y se emprende la acción apropiada.

Un programa bien escrito no tendrá más de una rutina ON ERROR. Dicha rutina no será capaz de tratar los errores de sintaxis, agotamiento de la memoria, desbordamiento de capacidad (*overflow*) de la pila, etc. Lo mejor que puede ofrecer esta facilidad es una ordenada suspensión del trabajo por parte del sistema, asegurando que todos los archivos queden cerrados (CLOSE) y que el usuario sepa exactamente qué es lo que ha sucedido.

Algunos errores, como por ejemplo una división por cero, que una rutina de este tipo podría manejar, en realidad se deben tratar de una forma diferente. Existen varias razones para ello:

- La instrucción ON ERROR GOTO y el subsiguiente salto hacia atrás al programa principal constituye una entrada y una salida extras hacia y desde una rutina. Ello viola el principio de la programación estructurada que establece que las rutinas han de tener un solo punto de entrada y un solo punto de salida.
- El lugar correcto para protegerse contra una división por cero es la propia rutina que realiza la división. Es una mala costumbre diseñar algoritmos que puedan romper el sistema. Si la verificación extra de errores implicada ralentiza el programa hasta un nivel inaceptable, se debe volver a diseñar la rutina de modo que no exista este riesgo.
- Las rutinas para tratamiento de errores se complican muy rápidamente si IF...THEN...ELSE enlaza con múltiples salidas. Se ven inevitablemente limitadas por la numeración de líneas del resto del programa y, por consiguiente, se deben reescribir siempre que se vuelva a diseñar cualquier rutina que las utilice. Éstas son particularmente difíciles de diseñar, comprobar y depurar, y cualquier error que exista en una rutina de esta clase puede introducir problemas de mayor envergadura al desviar el flujo de control de formas imprevistas.

Rompiendo barreras

El acceso ilícito a los ordenadores centrales utilizando máquinas personales y modems se conoce como "hacking" (asalto)

La película *Juegos de guerra* cautivó la imaginación de muchos usuarios de ordenadores personales. Utilizando un micro y un modem, el héroe se comunica ilícitamente con una sucesión de ordenadores para modificar los resultados de sus exámenes escolares, reservar billetes en líneas aéreas y cargar el software para los juegos más novedosos. Sin embargo, las cosas comienzan a ir mal cuando inadvertidamente consigue acceso al ordenador NORAD, responsable de la defensa aérea norteamericana, y está a punto de desencadenarse una guerra nuclear mundial. Una bonita historia para divertirse; ¿pero seguro que no es demasiado rebuscada?

Muchas "interrupciones" por ordenador de este tipo se han producido realmente, y el causante ha resultado ser, por lo general, un adolescente provisto de un micro personal y un modem. Las "víctimas" van desde potentes ordenadores centrales pertenecientes a universidades y grandes corporaciones, hasta servicios de tablones de comunicaciones dirigidos por entusiastas en microordenadores. Todos los ordenadores que permiten el acceso externo mediante el teléfono son vulnerables. En 1983, la realidad estuvo muy cerca de imitar a la ficción cuando se sospechó que dos "asaltantes" habían conseguido acceder al sistema de ordenador NORAD de Omaha (Nebraska).

Los dos adolescentes implicados eran de Los Ángeles y se las habían apañado para meterse en el Arpanet (la red secreta de ordenadores que posee el Departamento de Defensa de Estados Unidos). Utilizando un Commodore Vic-20 y un Tandy TRS-80, la pareja se las arregló para explorar el contenido de varios ordenadores conectados con el Arpanet, que suelen pertenecer a contratistas de defensa, organizaciones de investigación y universidades. A pesar de que no lograron obtener información secreta (el sistema se emplea básicamente para reunir datos científicos), la facilidad con que los dos adolescentes se "colaron" causó un enorme desconcierto en el Departamento de Defensa.

El motivo por el cual a los muchachos les resultó tan fácil tuvo más que ver con la indolencia humana que con cualquier fallo del ordenador. Todos los usuarios registrados del Arpanet poseen contraseñas; lamentablemente, éstas no se eligieron con mucha imaginación. En este caso, los muchachos supusieron que la Universidad de California de Berkeley podría ser usuario del Arpanet. A partir de este acierto, la contraseña "UCB" los introdujo en la red y entonces se vieron en libertad para acceder a cualquiera de los ordenadores conectados al Arpanet, uno de los cuales es el NORAD, de las oficinas centrales subterráneas de Omaha.

A pesar de que la sede central del NORAD está en el Arpanet, los ordenadores responsables de la verdadera defensa aérea no se encuentran allí. Están situados debajo de las montañas Cheyenne, en Colorado, y no están conectados a las líneas telefónicas públicas.

Puede que los ordenadores NORAD sean inmunes a la violación por parte de "asaltantes", pero otros no lo son. Otro incidente se produjo en julio de 1983, cuando un grupo de adolescentes de Milwaukee entró en más de 60 ordenadores pertenecientes a facultades, corporaciones y al Los Alamos National Laboratory, que se dedica a la fabricación de armas. Nuevamente, según las autoridades no se accedió a información secreta, sólo a registros, informes de rutina y mensajes. Se llamó al FBI para que averiguara por qué el grupo, cuyos miembros se autodenominaban los "414", había podido realizar tamaña proeza. Los 414 manifestaron que en ninguno de los ordenadores a los que habían llamado había mensajes de seguridad.

Éstos son apenas algunos de los casos a los que se ha dado publicidad. Muchos otros no llegaron a trascender a la opinión pública, porque son muy pocas las organizaciones que quieren que se sepa que en su ordenador central se ha infiltrado, supongamos, un joven de 17 años con un micro personal que vale 25 000 pesetas. Asimismo, muchas organizaciones no están al tanto de las infiltraciones: suele ser muy difícil saber si un usuario no registrado o impostor ha estado "en línea", aunque algunos de los "asaltantes" más descarados dejan mensajes desafiando a que los cojan y firman como "El aniquilador del sistema" o "El capitán Zap".

¿Cómo se realiza exactamente el *hacking* (asalto)? Todo "asaltante" potencial necesita un ordenador personal, un modem y una pizca de ingenio. El primer obstáculo es averiguar el número de teléfono de un ordenador. Para las redes de acceso público, como la Telecom Gold, de Gran Bretaña, o The Source, de Estados Unidos, ello no es difícil, ya que por lo general se anuncian profusamente.

Distribución ilegal

A pesar de que la empresa norteamericana Pepsi-Cola afirma no tener conocimiento del incidente, la misma fue víctima de un caso de "asalto" muy celebrado en los últimos años. De acuerdo a los informes, alguien en Estados Unidos accedió de forma ilegal a un ordenador de la empresa Pepsi-Cola en Canadá. Los "asaltantes" enviaron grandes partidas de Pepsi a puntos de destino preelegidos, con el objeto de desviar elevadas sumas de dinero hacia cuentas corrientes ilícitas

Ian McKinnell

En el caso de los ordenadores privados ya es un poco más difícil. Pero si se sabe aproximadamente dónde está situado el ordenador, entonces se puede utilizar la técnica empleada por el protagonista de *Juegos de guerra*: él programó a su micro para que marcara todos los números de teléfono posibles de su ciudad. Si respondía un ordenador (identificado por el silbido delator del tono de transmisión) entonces la máquina tomaba nota del número; pero si respondía una persona, el modem colgaba y marcaba otro número. Esto se puede realizar con un modem de llamada automática; ¡marcar los números a mano resultaría bastante pesado!

Una vez conectado con el ordenador, invariablemente se le solicita una contraseña. Algunas redes permiten un uso limitado si se digita "HUESPED" o "USUARIONUEVO", o si simplemente pulsa RETURN. Pero un verdadero "asaltante" probará y romperá la contraseña. Con frecuencia esto no es especialmente difícil, porque los usuarios tienden a hacer poco alarde de imaginación y utilizan nombres, como "SMITH", o palabras obvias como "SECRETO", o incluso simplemente "CONTRASEÑA". Del mismo modo, en el caso de las contraseñas compuestas por números, la gente tiende a escoger secuencias fáciles de recordar: por ejemplo, su fecha de nacimiento, como "090560". Muchos ordenadores son muy indulgentes y permiten varios intentos de contraseña antes de desconectarlo a uno. Aun así, normalmente se puede volver a marcar y seguir desde donde se quedó antes sin producir sospechas en el ordenador anfitrión.

Una vez en el sistema, la mayoría de los "asaltantes" se conforman con mirar los archivos, encontrar las páginas de juegos (si las hubiera) y "hablar" con otros "asaltantes" que también hayan conseguido meterse. Algunos de los más destructivos eliminan archivos, dejan mensajes obscenos e intentan "romper" todo el sistema; esto último puede tener consecuencias desastrosas para los usuarios.

Aun en los ordenadores centrales más sofisticados, es frecuente que los programadores dejen "puertas secretas" en el sistema de modo que, en un caso de emergencia, puedan evitar las medidas de protección e introducirse en el programa. Muchas veces las personas que operan el sistema desconocen la existencia de puertas de este tipo.

Usted habrá observado que la mayoría de los casos que hemos reseñado se refieren a ordenadores de universidades. Ello se debe a que esta clase de ordenadores, aparte de tener un acceso por línea exterior, por lo general operan con una política de acceso libre. Con miles de usuarios y muchos lugares remotos, ésta es la forma más práctica de llevar un sistema de este tipo. Lamentablemente, también son presa fácil para los "asaltantes" que deseen entrar en ellos, y una vez allí pueden "ir saltando" de un ordenador a otro actuando como si fueran usuarios legítimos. Un estudiante del *campus* de San Jose State descubrió un agujero en un bucle del programa *Talk* del ordenador de la universidad, que permite que los estudiantes "hablen" con otras ciudades universitarias de la California State University. El estudiante consiguió superar la restricción local y logró comunicarse con ordenadores de Suecia, Irán y China, así como de diversos lugares de Estados Unidos. La factura del teléfono ascendió a más de $ 10 000.

¿Qué persiguen los "asaltantes" al actuar de esta manera? Muchos de ellos poseen un código de conducta extraoficial y no borran archivos ni dejan mensajes obscenos. Para ellos la emoción reside simplemente en romper los códigos.

Durante mucho tiempo los bancos han sido objeto de delitos por ordenador, pero hasta hace poco todos ellos se hacían "de puertas adentro", es decir, por poner un ejemplo, empleados deshonestos que transferían dinero a cuentas falsas. Las estimaciones sobre el fraude por ordenador varían, sólo en Gran Bretaña, desde £30 millones hasta más de £2 500 millones al año. Como es fácilmente comprensible, es poco frecuente que los bancos y las empresas admitan con carácter público que han sido víctimas de un fraude por ordenador y, por tanto, es difícil efectuar estimaciones exactas.

El incremento en el número de propietarios de ordenadores personales y la creciente cantidad de redes de ordenadores que utilizan las líneas telefónicas significan que los delitos por ordenador sólo pueden seguir una espiral ascendente.

Ensayo y error

Ian McKinnell

El procedimiento que utiliza un "asaltante" supone un gran trabajo de ensayo y error que en muy contadas ocasiones conduce a lograr introducirse en el sistema sin autorización. Incluso aunque un "asaltante" sea capaz de localizar un sistema determinado, a menudo se encontrará con un diálogo de esta naturaleza cuando intente colarse en el sistema:

```
CONEXION QX001001 14, 32
        12/7/84
>                          Pulsar return: se solicita ID del usuario
USER?                      No hay ayuda
> HELP                     Primer intento de ID válida
USER?                      ID no válida para acceso
> $ OFF
USER?                      Segundo intento de ID válida
> BN001                    ID aceptada: se solicita la contraseña
PASSWORD?                  Primer intento
> HUESPED                  No se acepta: 2.ª solicitud
PASSWORD?                  Segundo intento
> NUEVOUSUARIO             No se acepta: 3.ª solicitud
PASSWORD?                  Intento desesperado
> QWERTY                   El usuario queda desconectado después del tercer intento fallido por descubrir contraseña
LOGOFF 15.13 CONNECT
        TIME = 0.13 MINS
```

En algunos casos, ya sea por elucubración o por pura suerte, una persona logra descubrir una contraseña de gran prioridad válida y se mete en el sistema. El siguiente diálogo imaginario describe cómo un usuario de esta clase podría acceder a información confidencial relativa a un usuario legal:

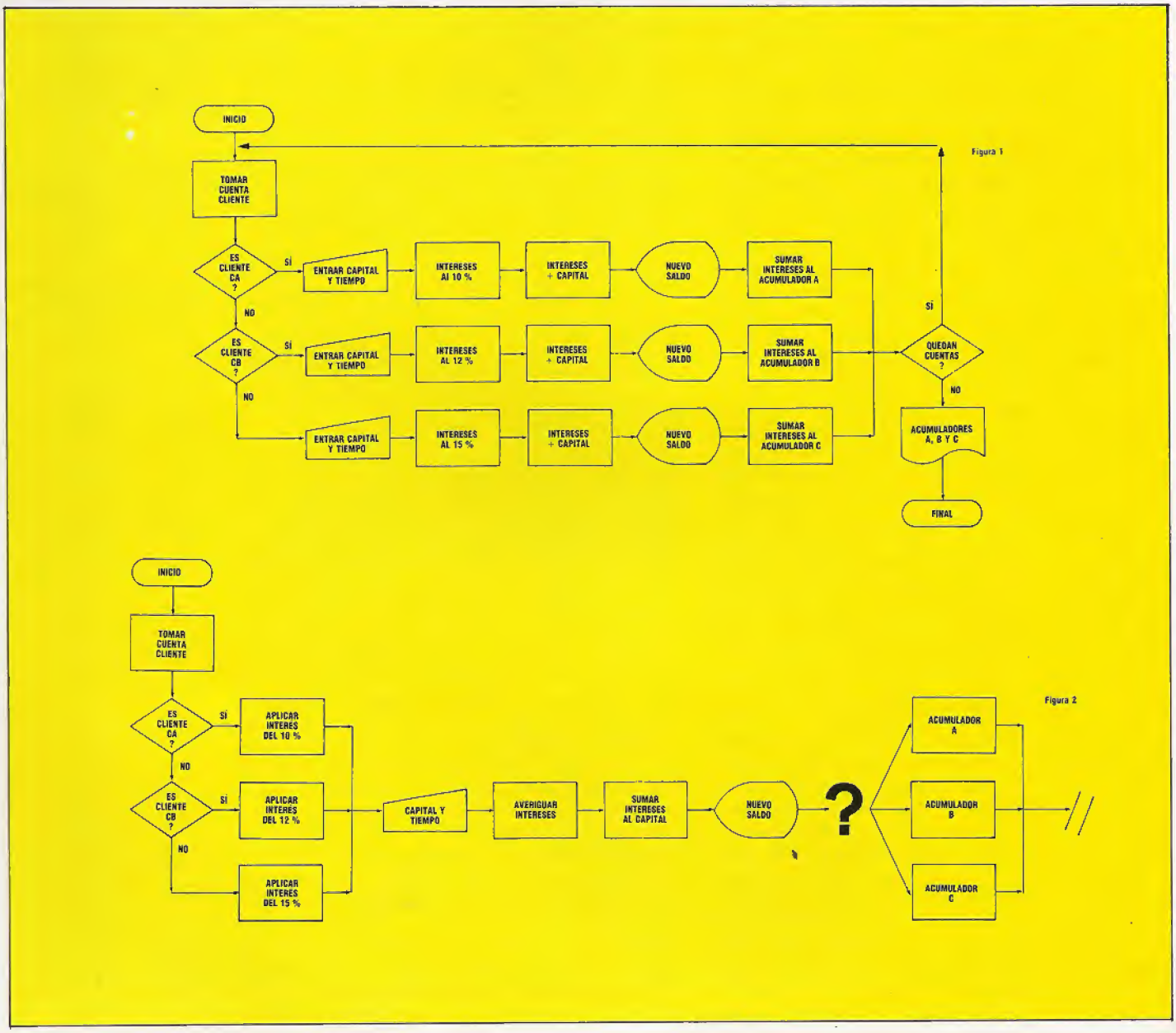
```
CONNECT BYF990
        15.14.02 12/07/84
>                          Pulsar retorno
USER?                      ID válida: solicita contraseña
> BN001                    Primer intento: operador sistema
PASSWORD?
> SYSOP
LOGON 15.15.07 12/07/84
HOST: BYF990/SPYLOM
USER: BN001                Número de serie del software
SERIAL NO: ZA180-7         Paso libre a los archivos
PRIORITY: SUPERUSUARIO     Los usuarios se pueden considerar inactivos si no usan el sistema regularmente
STATUS: ACTIVE             Confirmación
YOU ARE SYSOP              Lista todos los usuarios
7 USER(S)                  Instrucción reservada para operador del sistema
APP01 APP02 BYF7 BTY04     Usuario válido eliminado
        BZX88 BZX02 SYSOP
> REMOVE ARP01
USER(S) ARP01
        DISCONNECTED(S)
> WHO IS BTY04
BTY04      ROSA RUPEREZ,
        BTY LOPP, 742
        SILICON DV
HERTS 07662-093164
```

Acumuladores

Profundicemos un poco en la utilización de acumuladores mediante un ejemplo práctico

Los clientes de una caja de ahorros o de un banco se agrupan en tres categorías según su tipo de cuenta: CA, CB y CC, y a cada una de estas categorías se aplican intereses diferentes, del 10, 12 y 15 %, respectivamente. Para cada uno de los clientes deben calcularse los intereses correspondientes a su saldo y visualizar la nueva cantidad una vez incrementada con los citados intereses. De igual manera, cuando ya se haya procesado la totalidad de clientes, se imprimirá el total por categoría de los intereses entregados.

Con lo aprendido hasta el momento, se conseguiría la representación de la figura 1, ante la imposibilidad de poder agrupar en un punto determinado de la secuencia las operaciones repetitivas. En caso de no hacerse así, no se podría después diferenciar en qué registro deben acumularse los intereses como muestra la figura 2. Al llegar al punto en el que figura el interrogante, el proceso no continuaría por falta de datos, o si lo hiciera sería por defecto, en cuyo caso el resultado sería erróneamente dirigido siempre al mismo acumulador.



Un nuevo hito

Este micro lanzado recientemente es la primera máquina personal que proporciona una unidad de disco incorporada

La etiqueta del precio del Einstein indica que gustará principalmente al usuario personal "serio". Claro está que se puede utilizar para juegos, pero en este campo ofrece pocas ventajas en relación a máquinas que valen la cuarta parte que ésta. Su competidor más cercano, en cuanto a precio y rendimiento, es el BBC Micro, pero los 80 Kbytes de RAM del Einstein le sacan ventaja a los más bien escasos 32 Kbytes que ofrece Acorn.

La unidad de disco está montada en un panel justo arriba del teclado en la carcasa del Einstein, que es inusualmente grande. Esta carcasa tiene la solidez suficiente como para aguantar el peso de una pantalla o un televisor, de manera que el sistema completo no ocupa mucho espacio.

La principal ventaja de la unidad de disco integral probablemente sea la disponibilidad de software. Si bien existen otras máquinas personales, como el Commodore 64, que se podrían equipar con unidades de disco, el hecho de que hayan sido diseñadas pensando en grabadoras de cassette significa que la mayor parte del software se fabrica en cassette y que, en consecuencia, los propietarios de unidades de disco no podrán aprovechar al máximo el superior soporte de almacenamiento de que disponen. La unidad incorporada del Einstein asegura que todo el software se suministrará en disco desde el principio. El empleo de discos permite cargar programas y datos rápidamente y posibilita la utilización de archivos de acceso directo en vez de los archivos de acceso secuencial a los que se ven limitadas las máquinas basadas en cassette. En cada cara del disco de 3 pulgadas se pueden almacenar 190 Kbytes de datos, pero el Einstein sólo se puede utilizar una cara a la vez. En la carcasa se puede instalar una segunda unidad de disco, y en una interface situada en la parte posterior de la máquina se pueden enchufar otras dos unidades.

Para controlar el empleo de la unidad de disco, el Einstein posee su propio sistema operativo de disco (DOS: *disk operating system*). Éste posee muchas similitudes con el CP/M estándar que utilizan muchas máquinas de oficina, y Tatung confía en que las casas de software convertirán los programas CP/M para su ejecución en el Einstein. El sistema operativo y el BASIC Einstein no están contenidos en ROM, como ocurre generalmente, sino que se deben cargar desde disco cada vez que se conecta la máquina. Ello ofrece dos ventajas fundamentales: como el BASIC se carga sólo cuando se lo necesita, otros lenguajes de programación o programas en lenguaje máquina pueden utilizar el espacio completo de RAM, y tanto el sistema operativo como el BASIC se pueden actualizar fácilmente mediante la adquisición de un disco que contenga las nuevas versiones. Tatung tiene planeado ofrecer el lenguaje DR LOGO en el disco que se suministra de forma gratuita con la máquina; pero este obsequio no es todo lo generoso que puede parecer, porque el manual de LOGO se venderá aparte. Una vez cargado el BASIC desde el disco, el Einstein posee unos valiosos 43 Kbytes de RAM disponibles para el usuario, que es más de lo que ofrece cualquier otra máquina personal. Esto es factible debido a que la RAM del Einstein contiene 16 Kbytes de memoria separados que se controlan mediante el chip de gráficos y se usan para la visualización en pantalla.

El BASIC Einstein parece ser una mezcla de BASIC BBC y Microsoft Extended BASIC (como el que utilizan las máquinas MSX japonesas). Incluye instrucciones para renumerar programas y para producir números de línea de forma automática, lo que facilita la entrada de programas. Un editor de pantalla completo ofrece la posibilidad de efectuar cambios en cualquier lugar de la visualización en pantalla. Las instrucciones para gráficos permiten dibujar líneas, círculos y elipses, así como rellenar

Chris Stevens

Sistema serio
El ordenador Einstein, fabricado por Tatung (antiguamente Decca). La máquina está destinada al usuario personal serio y viene equipada con una unidad de disco. Es más grande que la mayoría de las máquinas personales, lo que permite apoyar la pantalla sobre la carcasa

formas con color sólido. Los gráficos tienen una resolución máxima de 256 × 192 pixels y hay 15 colores disponibles, si bien no se pueden utilizar más de dos para cada fila de ocho pixels. El usuario puede definir hasta 32 caracteres de sprites; las instrucciones para controlarlos están incluidas en el BASIC, lo que permite escribir excelentes programas para juegos de acción muy rápida. Se suministra un programa monitor en ROM para hacer que la programación en lenguaje máquina resulte más sencilla.

Asimismo, el chip de gráficos limita la visualización a 40 caracteres a lo ancho. Existen planes para poner en el mercado un accesorio que permita reproducir en el Einstein la visualización de 80 caracteres que exigen muchos programas CP/M; la pantalla de 80 columnas será sólo monocromática, pero hay una versión en color prevista para 1985.

La calidad de sonido del Einstein es buena, con la salida dirigida a un gran altavoz situado encima del teclado. Se proporciona un control de volumen; las instrucciones del BASIC para generar sonido son amplias y fáciles de emplear.

Existen ocho teclas de función; éstas se pueden programar para producir palabras e instrucciones comúnmente utilizadas, y una banda plástica transparente permite fijar etiquetas encima de cada tecla, al estilo del BBC. El teclado está bien construido y la mecanografía al tacto no debería suponer ningún problema, pero Tatung sólo ha suministrado dos teclas de cursor en vez de las cuatro habituales. Esto significa que las teclas del cursor se deben utilizar junto con la tecla de cambio para producir el movimiento en dos de las cuatro direcciones, lo que resulta incomprensible en una máquina tan cara. Se puede producir un juego de caracteres para gráficos desde el teclado si se mantiene pulsada la tecla para gráficos, pero los mismos tienen un empleo limitado.

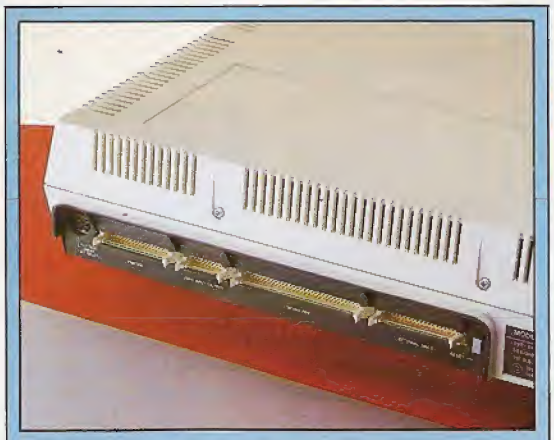
En cuanto a cantidad de interfaces disponibles, el único competidor del Einstein es el BBC Micro. Éstas incluyen una interface Centronics estándar para conexión con impresora; un conector RS232 para utilizar con impresoras, modems y otros accesorios; un conector para una pantalla a color RGB; una salida para visualización en televisor y un par de conectores para palanca. A los conectores para palanca también se les puede dar una utilidad más seria, dado que son del tipo convertidor de analógico a digital que permite medir voltajes eléctricos. Los dos conectores proporcionan cuatro canales de A a D; éstos se complementan con una puerta para el usuario de ocho bits que puede recibir y producir señales digitales desde y hacia otros componentes del equipo. Esta combinación de puertas para el usuario y de A a D hace que el Einstein sea ideal para control en robótica y aplicaciones científicas.

La futura ampliación es factible en función del "Pipe" (similar en concepto al "Tube" del BBC Micro), que permitirá instalar diversos accesorios. Un conector de ROM dentro de la máquina permite ampliar a 32 Kbytes los ocho Kbytes de ROM que tiene como estándar.

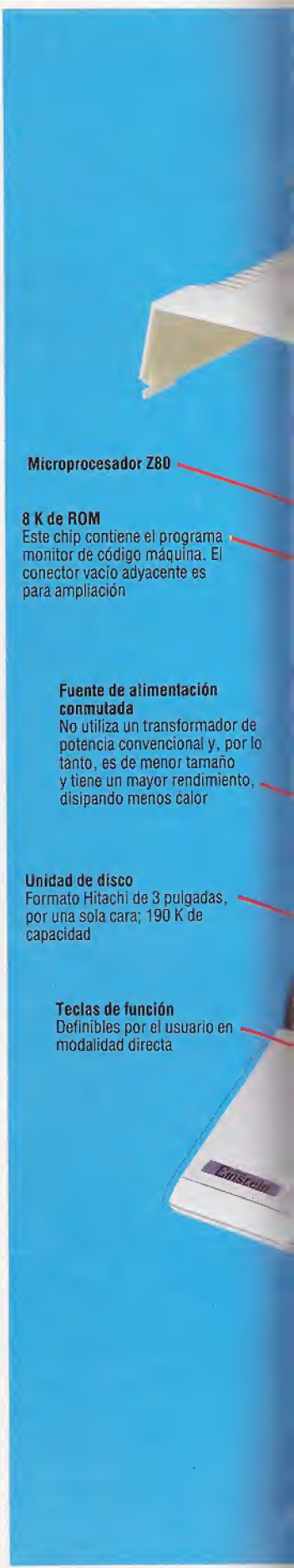
El Einstein, fabricado en Gran Bretaña, sin duda alguna bien vale su precio; pero la realidad es que son pocos los usuarios que se pueden permitir gastarse el equivalente de £500 en un ordenador personal. Todavía es muy escaso el software disponible, y la situación no cambiará a menos que la máquina alcance un volumen de ventas considerable.

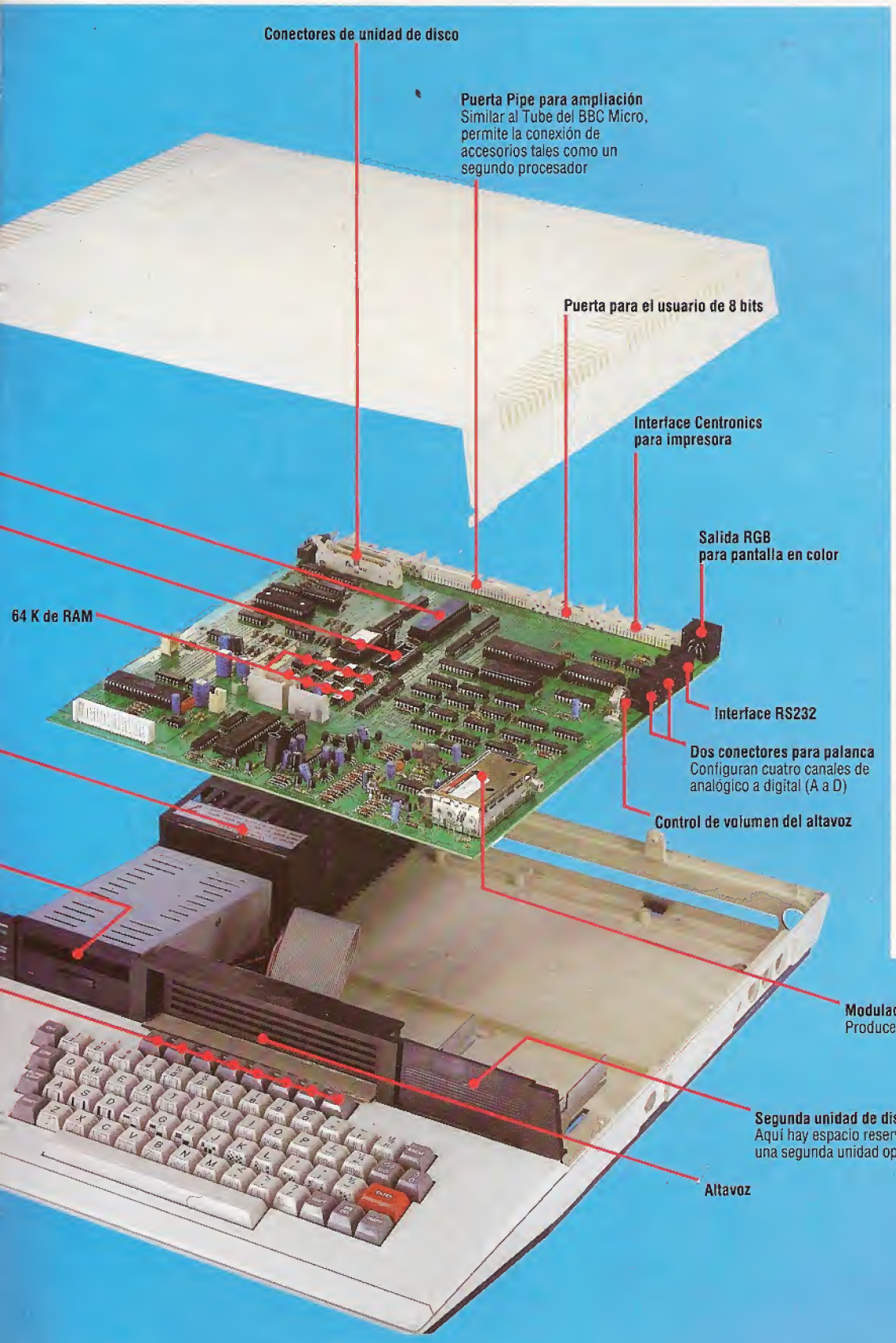
Opciones de pantalla
La pantalla Tatung acepta entradas RGB o YUV; ésta última es el sistema propio de Tatung, supuestamente superior. Dado que la salida YUV incluye una línea de video compuesto, con ella se puede utilizar cualquier pantalla

Unidad de disco Hitachi
La unidad de disco responde al sistema Hitachi de 3 pulgadas, que se está haciendo cada vez más popular entre los microordenadores. Los discos pueden almacenar hasta 190 Kbytes, y si bien la unidad sólo lee una cara, los discos se pueden dar vuelta de forma manual para utilizar las dos caras



Interface Einstein
El Einstein está bien dotado de interfaces, incluyendo el "Pipe" Tatung, una puerta de propósito general. También hay una interface para unidades de disco adicionales





TATUNG EINSTEIN

DIMENSIONES
510 x 430 x 105 mm

CPU
Z80

MEMORIA
80 K de RAM, incluyendo 16 K de memoria de pantalla. Ocho K de ROM, ampliables a 32 K

PANTALLA
Texto: 32 columnas x 24 filas/40 columnas x 24 filas. Gráficos: 256 x 192 pixels con 15 colores

INTERFACES
Centronics, RS232, dos palancas de mando (de A a D), RGB, "Pipe", puerta para el usuario y unidad de disco externa

LENGUAJES DISPONIBLES
Se proporcionan BASIC y DR LOGO. Se venden PASCAL y FORTH

TECLADO
67 teclas tipo máquina de escribir, trazado QWERTY. Incluye ocho teclas de función programables

DOCUMENTACION
Tres manuales, incluyendo uno de BASIC pobre

VENTAJAS
Unidad de disco incorporada, muchas interfaces, enorme memoria para el usuario, buen BASIC, gráficos sprite

DESVENTAJAS
Poco software disponible. Precio de venta al público elevado

Chris Stevens

Explosión final

Concluimos nuestro juego con un listado completo del programa y ofrecemos las líneas alternativas para ejecutarlo en el Electron

La modalidad de gráficos 7 del BBC Micro, también conocida como modalidad de teletexto, posee varias características de las que no dispone en ninguna otra modalidad. Éstas se utilizan para la visualización de información transmitida desde fuentes exteriores, como el Micronet, al que se puede acceder utilizando el ordenador y una línea telefónica normal. Las características para gráficos adicionales que se pueden obtener mediante el empleo de la modalidad 7 permiten producir atractivas visualizaciones con letras mediante unas sencillas y breves instrucciones; por consiguiente, esta modalidad es una alternativa ideal para nuestra pantalla de "final del juego".

Mediante la utilización de códigos de control CHR$ en sentencias PRINT podemos controlar los colores del texto y del fondo, crear texto "intermitente" y producir caracteres de doble altura. Podemos emplear la función TAB de la forma normal para posicionar el texto en la pantalla de 40 por 25 caracteres. Hay siete colores disponibles y éstos se pueden seleccionar mediante los siguientes códigos de control:

129	rojo
130	verde
131	amarillo
132	azul
133	magenta
134	cyan
135	blanco

Siempre que se emplea la modalidad 7, el texto se visualiza en blanco sobre fondo negro. El color del texto se puede modificar en cualquier punto de una sentencia PRINT mediante la inserción de un código de control. Por ejemplo, las tres palabras de la frase de la siguiente línea se imprimirán en rojo, blanco y azul, respectivamente:

```
PRINT CHR$(129)"JUEGO";CHR$(135);"SIN";
      CHR$(132);"FRONTERAS"
```

Es importante, sin embargo, comprender que cuando se imprima otra línea se restaurará el color por defecto (blanco). Por lo tanto debemos utilizar códigos de control en cada nueva línea, aun cuando deseemos seguir imprimiendo en el mismo color.

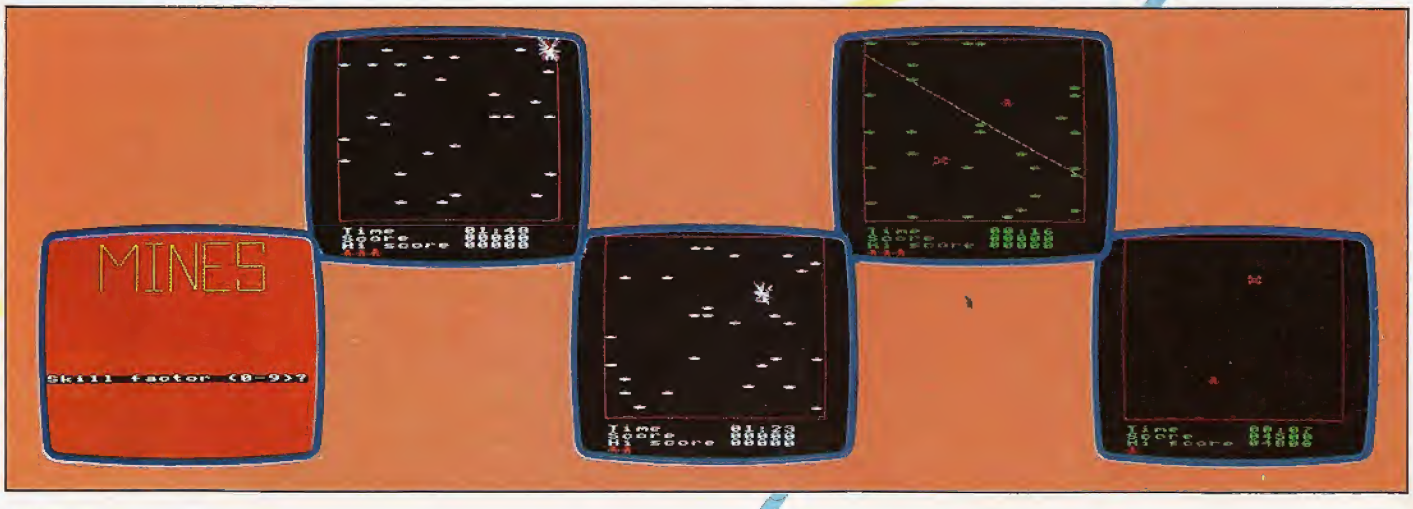
Además del color del texto, también podemos seleccionar los colores de fondo. CHR$(157), seguido del color que deseamos para el fondo, nos permite hacer esto. Por ejemplo, para producir letras azules sobre fondo blanco se utiliza la siguiente combinación de códigos de control:

```
PRINT CHR$(132);CHR$(157);CHR$(135);"EN LA
      MARINA"
```

El primer código de control especifica el color del texto; el segundo y el tercero definen el fondo. Se puede hacer que tanto el texto como el fondo sean intermitentes utilizando un código de control CHR$(136) inmediatamente antes del código de color. CHR$(137) desactiva este efecto. Por ejemplo, podemos hacer que las letras azules del ejemplo anterior sean intermitentes mediante:

```
PRINT CHR$(136);CHR$(132);CHR$(157);
      CHR$(135);"EN LA MARINA"
```

La característica más notable de la modalidad 7 es su capacidad para producir caracteres de doble altura. CHR$(141) nos permite hacerlo, pero también debemos imprimir (PRINT) la misma línea dos veces para conseguir el efecto correcto. A los caracteres de doble altura les podemos aplicar todos los otros efectos. Por ejemplo, para producir caracteres azu-



¡Peligro!: campo minado
Estos fotogramas de diversos momentos del juego muestran las minas, el ayudante, el disparo del francotirador, una explosión y las visualizaciones del marcador y los títulos

les intermitentes de doble altura sobre un fondo blanco fijo, usamos las siguientes líneas de código:

```
10 MODE 7
20 PRINT CHR$(141);CHR$(136);CHR$(132);
   CHR$(157);CHR$(135);"EN LA MARINA"
30 PRINT CHR$(141);CHR$(136);CHR$(132);
   CHR$(157);CHR$(135);"EN LA MARINA"
```

Sin embargo, todos estos códigos de control ocupan muchísimo espacio de programa y su digitación lleva mucho tiempo. Un método alternativo consiste en encadenar varios códigos entre sí formando una serie única que se pueda utilizar en las sentencias PRINT. Por ejemplo, si necesitáramos realizar muchos PRINT empleando caracteres rojos sobre fondo amarillo, podríamos empezar por crear una serie (rojo$) que se pudiera luego utilizar en cada sentencia PRINT que requiriera este efecto:

```
10 MODE 7
20 rojo$ = CHR$(129) + CHR$(157) + CHR$(131)
30 PRINTrojo$;"JUEGO TERMINADO"
40 PRINTrojo$;"SU MARCADOR"
```

El procedimiento para el "final del juego"

Examinemos ahora más de cerca cómo usamos estos efectos para la pantalla de final del juego:

```
2110 DEF PROCfinal_juego
2120 IF marcador$ > max_marcador$ THEN max_marcador$ = marcador$
2130 rojo$ = CHR$(129) + CHR$(157) + CHR$(131)
2140 juego$ = "FIN JUEGO"
2150 PRINTTAB(0,5)rojo$;CHR$(141);CHR$(136);TAB(12);juego$
2160 PRINTrojo$;CHR$(141);CHR$(136);TAB(12);juego$
2170 PRINT:PRINTrojo$;"Su marcador";TAB(30);marcador$
2180 PRINT:PRINTrojo$;"Max marcador";TAB(30);max_marcador$
2190 PRINT:PRINTrojo$;"Tiempo";TAB(30);tiempo$
2200 azul$ = CHR$(132) + CHR$(157) + CHR$(134)
2210 go$="OTRA PARTIDA S/N?"
2220 PRINT:PRINT
2230 PRINTazul$;CHR$(141);CHR$(136);TAB(5);go$
2240 PRINTazul$;CHR$(141);CHR$(136);TAB(5);go$
2250 REM ** RESPUESTA ? **
2260 *FX 15,1
2270 respuesta$ = INKEY$(0)
2280 IF GET$ = "N" THEN flag_final = 1
2290 ENDPROC
```

La línea 2120 comprueba si el marcador del juego que acaba de concluir es mayor que el máximo marcador anterior y, caso de ser necesario, actualiza el marcador de puntuación máxima.

El mensaje FIN JUEGO se imprime luego en caracteres intermitentes rojos de doble altura sobre fondo amarillo (líneas 2130 a 2160), y se visualizan los detalles de los marcadores y el tiempo (líneas 2170 a 2190). Se le pregunta entonces al jugador si desea jugar otra vez. Si la respuesta es N, entonces se pone una variable (flag-final) en uno.

Observe que durante este procedimiento no se ha establecido la modalidad 7. Ello se debe a que el BBC Micro no permite efectuar un cambio de modalidad dentro de un procedimiento. Si se intentara, se produciría un mensaje de error BAD MODE (modalidad incorrecta). Debemos, en cambio, establecer la modalidad 7 en el corto programa principal que llama a los procedimientos. Para completar el programa se deben agregar las siguientes líneas. Ahora todo el programa de llamada completo se ha colocado en un bucle REPEAT...UNTIL, que se repetirá hasta que flag-final se ponga a uno.

```
1100 REPEAT

1200 MODE7
1210 REM ** APAGAR CURSOR **
1220 VDU23,1,0;0;0;0;
1230 PROCfinal_juego
1240 UNTIL flag-final = 1
1250 CLS
1260 END
```

La alternativa de Electron

Los usuarios del Electron deben haber ido leyendo nuestro análisis de la modalidad 7 con cierta preocupación, porque el Electron no dispone de la misma. Como alternativa, hemos preparado un procedimiento diferente que utiliza la modalidad 5 para la pantalla del final del juego. Omita la línea 1200 del programa de llamada que acabamos de proporcionar e introduzca este procedimiento en lugar del de final del juego para el BBC:

```
> L.2100,2300
 2100 DEF PROCfinal_juego
 2110 IF marcador$ > max_marcador$ THEN max_marcador$ = marcador$
 2120 REM ASEGURAR FONDO AMARILLO
 2130 VDU19,130,3,0,0,0
 2140 GCOL0,130:REM COLOREAR PANTALLA
 2150 COLOUR1:COLOUR130:REM ESTABLECER COLORES TEXTO
 2160 juego$ = "FIN JUEGO"
 2170 PRINTTAB(2,4);juego$
 2180 COLOUR0
 2190 PRINTTAB(0,8);"Su marcador";TAB(15);marcador$
 2200 PRINT:PRINT"Max marcador";TAB(15);max_marcador$
 2210 PRINT:PRINT"Tiempo";TAB(15);tiempo$
 2220 go$ = "OTRA PARTIDA S/N?"
 2230 REM CAMBIAR COL3 A AMARILLO/AZUL INTERMITENTE
 2240 VDU19,3,11,0,0,0
 2250 COLOUR3
 2260 PRINT:PRINT
 2265 PRINTTAB(2)go$
 2270 REM ** RESPUESTA  ? **
 2275 *FX 15,1
 2280 respuesta$ = INKEY$(0)
 2285 IF GET$ = "N" THEN flag_final = 1
 2290 VDU 20:REM RESTAURAR COLORES POR DEFECTO
 2300 ENDPROC
```

El listado final

```
1000 REM **************************
1010 REM **                      **
1020 REM **        MINAS         **
1030 REM **                      **
1040 REM **************************
1050 :
1060 max_marcador$ = "00000"
1070 flag_final = 0
1080 :
1090 REM **** PROGRAMA PRINCIPAL ****
1100 REPEAT
1110 MODE5
1120 REM ** APAGAR CURSOR **
1130 VDU23;8202;0;0;0;
1140 PROCpagina_titulos
1150 CLS
1160 PROCpreparacion
1170 :
1180 PROCbucle
1190 :
1200 MODE7
1210 REM ** APAGAR CURSOR **
1220 VDU23,1,0;0;0;0;
1230 PROCfinal_juego
1240 UNTIL flag_final = 1
1250 CLS
1260 END
1270 :
1280 :
1290 REM **** DEFINICION DE PROCEDIMIENTOS ****
1300 DEF PROCpagina_titulos
1310 GCOL 0,129
1320 CLG
1330 GCOL 3,3
1340 PROCmusica
1350 Y = 100:X = 0
1360 REPEAT
1370 X = X + 20:Y = Y + 50
1380 FOR I = 1 TO 2
1390 PROCminas
1400 NEXT I
1410 UNTIL Y > 700
1420 :
1430 PROCminas
1440 PRINTTAB(0,20)"Factor de destreza (0-9)?"
1450 PROCmusica
1460 REPEAT
1470 destreza = GET-48
1480 UNTIL destreza > -1 AND destreza < 10
1490 ENDPROC
1500 :
1510 DEF PROCminas
1520 PLOT4,X,Y
1530 REM ** LETRA M **
1540 PLOT1,0,200
1550 PLOT1,80,-100
1560 PLOT1,80,100
1570 PLOT1,0,-200
1580 REM ** LETRA I **
1590 PLOT0,40,0
1600 PLOT1,80,0
1610 PLOT0,-40,0
1620 PLOT1,0,200
1630 PLOT0,-40,0
1640 PLOT1,80,0
1650 REM ** LETRA N **
1660 PLOT0,40,-200
1670 PLOT1,0,200
1680 PLOT1,120,-200
1690 PLOT1,0,200
1700 REM ** LETRA E **
1710 PLOT0,160,0
1720 PLOT1,-120,0
```

Listado final
Este listado muestra claramente la ortografía de los nombres de variables y procedimientos (como max_marcador$ y PROCfinal_juego), que incluyen el carácter de subrayado, "_". Los programadores del BBC Micro ya estarán familiarizados con su utilización como carácter legal de separación; no se lo debe confundir con el guión

```
1730 PLOT1,0,-200
1740 PLOT1,120,0
1750 PLOT0,-40,100
1760 PLOT1,-80,0
1770 REM ** LETRA S **
1780 PLOT0,280,60
1790 PLOT 1,0,40
1800 PLOT1,-120,0
1810 PLOT1,0,-100
1820 PLOT1,120,0
1830 PLOT1,0,-100
1840 PLOT1,-120,0
1850 PLOT1,0,40
1860 ENDPROC
1870 :
1880 DEF PROCpreparacion
1890 COLOUR2
1900 flag_final = 0
1910 PROCinicializar_variables
1920 PROCdefinir_caracteres
1930 factor = destreza*3 + 30
1940 PROCcolocar-minas(factor)
1950 PROCtrazar_borde
1960 PROCestablecer_tiempo
1970 PROCestablecer_marcador
1980 PROCestablecer_hombres
1990 PROCsituar_sujetos
2000 ENDPROC
2010 :
2020 DEF PROCbucle
2030 REPEAT
2040 PROCactualizar_tiempo
2050 PROCleer_teclado
2060 rand = RND(50-destreza)
2070 IF rand = 1 THEN PROCfrancotirador
2080 UNTIL TIME > 12099 OR flag_final = 1
2090 ENDPROC
2100 :
2110 DEF PROCfinal_juego
2120 IF marcador$ > max_marcador$ THEN max_marcador$ = marcador$
2130 rojo$ = CHR$(129) + CHR$(157) + CHR$(131)
2140 juego$ = "FIN JUEGO"
2150 PRINTTAB(0,5)rojo$;CHR$(141);CHR$(136);TAB(12);juego$
2160 PRINTrojo$:CHR$(141);CHR$(136);TAB(12),juego$
2170 PRINT:PRINTrojo$;"Su marcador";TAB(30);macador$
2180 PRINT:PRINTrojo$;"Max marcador";TAB(30);max_marcador$
2190 PRINT:PRINTrojo$;"Tiempo";TAB(30);tiempo$
2200 azul$ = CHR$(132) + CHR$(157) + CHR$(134)
2210 go$ = "OTRA PARTIDA S/N?"
2220 PRINT:PRINT
2230 PRINTazul$;CHR$(141);CHR$(136);TAB(5);go$
2240 PRINTazul$;CHR$(141);CHR$(136);TAB(5);go$
2250 REM ** RESPUESTA ? **
2260 *FX 15,1
2270 respuesta$ = INKEY$(0)
2280 IF GET$ = "N" THEN flag_final = 1
2290 ENDPROC
2300 :
2310 REM **** PROCEDIMIENTOS NIVEL 2 ****
2320 DEF PROCinicializar_variables
2330 xdet = 2:ydet = 25:xhom = 17:yhom = 1
2340 xcomienzo = 120:xfinal = 1144
2350 cero$ = "000000"
2360 ENDPROC
2370 :
2380 DEF PROCdefinir_caracteres
2390 REM ** MINAS **
2400 VDU23,224,0,0,56,254,254,124,0,0
2410 REM ** DETECTOR DE MINAS **
2420 VDU23,225,231,195,189,36,36,189,195,231
2430 REM ** AYUDANTE **
2440 VDU23,226,56,56,16,124,186,170,40,108
2450 ENDPROC
2460 :
2470 DEF PROCtrazar_borde
2480 GCOL 0,1
2490 MOVE 120,188
2500 DRAW 120,992
2510 DRAW 1152,992
2520 DRAW 1152,188
2530 DRAW 120,188
2540 ENDPROC
2550 :
2560 DEF PROCcolocar_minas(numero_minas)
2570 REM ** CAMBIAR COLOUR 2 A VERDE **
2580 VDU19,2,2,0,0,0
2590 FOR I = 1 TO numero_minas
2600 PRINTTAB(RND(16) + 1,RND(25));CHR$(224)
2610 NEXT I
2620 ENDPROC
2630 :
2640 DEF PROCestablecer_tiempo
2650 PRINTTAB(2,27)"Tiempo      02:00"
2660 TIME = 0
2670 ENDPROC
2680 :
2690 DEF PROCestablecer_hombres
2700 hombres$ = CHR$(226) + CHR$(226) + CHR$(226)
2710 contador = 1
2720 COLOUR 1
2730 PRINTTAB(2,30);hombres$
2740 COLOUR 2
2750 ENDPROC
2760 :
2770 DEF PROCestablecer_marcador
2780 marcador = 0:marcador$ = "00000"
2790 PRINTTAB(2,28)"Marcador      00000"
2800 PRINTTAB(2,29)"Max marcador";max_marcador$
2810 ENDPROC
2820 :
2830 DEF PROCsituar_sujetos
2840 COLOUR 1
2850 PRINTTAB(xdet,ydet);CHR$(225)
2860 PRINTTAB(xhom,yhom);CHR$(226)
2870 COLOUR 2
2880 ENDPROC
2890 :
2900 DEF PROCactualizar_tiempo
2910 seg$ = STR$(((12100-TIME) DIV 100)MOD 60)
2920 min$ = STR$(((12100-TIME)DIV 6000)MOD 60)
2930 REM ** AGREGAR CEROS POR DELANTE **
2940 seg$ = LEFT$(cero$,2-LEN(seg$)) + seg$
2950 min$ = LEFT$(cero$,2-LEN(min$)) + min$
2960 tiempo$ = min$+":"+ seg$
2970 PRINTTAB(11,27);tiempo$
2980 ENDPROC
2990 :
3000 DEF PROCleer_teclado
3010 REM ** ARRIBA ? **
3020 IF INKEY(-58) = -1 THEN PROCmover(0,-1)
3030 REM ** ABAJO ? **
3040 IF INKEY(-42) = -1 THEN PROCmover(0,1)
3050 REM ** DERECHA ? **
3060 IF INKEY(-122) = -1 THEN PROCmover(1,0)
3070 REM ** IZQUIERDA ? **
3080 IF INKEY (-26) = -1 THEN PROCmover(-1,0)
3090 ENDPROC
3100 :
3110 DEF PROCfrancotirador
3120 ycomienzo = RND(750) + 220
3130 yfinal = RND(750)+220
3140 dx = 32:dy = (yfinal-ycomienzo)/32
3150 GCOL 3,3
3160 PROClinea
3170 IF POINT(x,y) = 1 THEN PROCexplotar(x,y) ELSE PROClinea
3180 ENDPROC
3190 :
3200 REM **** PROCEDIMIENTOS NIVEL 3 ****
3210 :
3220 DEF PROCmover(delta_x,delta_y)
3230 REM ** BORRAR POSICIONES ANTIGUAS **
3240 COLOUR 1
3250 PRINTTAB(xdet,ydet);" "
3260 PRINTTAB(xhom,yhom);" "
3270 REM ** MOVER DETECTOR **
3280 xdet = xdet + delta_x
3290 ydet = ydet + delta_y
3300 REM ** COMPROBAR LIMITES **
3310 IF xdet > 17 THEN xdet = 17
3320 IF ydet > 25 THEN ydet = 25
3330 IF xdet < 2 THEN xdet = 2
3340 IF ydet < 1 THEN ydet = 1
3350 REM ** CALCULAR COORDENADAS DEL HOMBRE **
3360 xhom = 19-xdet
3370 yhom = 26-ydet
3380 PROCconvertir(xhom,yhom)
3390 IF POINT(xgraf,ygraf) = 2 THEN PROCexplotar(xgraf,ygraf)
3400 PROCconvertir(xdet,ydet)
3410 IF POINT(xgraf,ygraf) = 2 THEN PROCdescubierta_mina
3420 PROCsituar_sujetos
3430 ENDPROC
3440 :
3450 DEF PROClinea
3460 SOUND0,-8,4,5
3470 x = xcomienzo:y = ycomienzo
3480 MOVE x,y
3490 REPEAT
3500 DRAW x,y
3510 x = x + dx:y = y + dy
3520 UNTIL x > xfinal OR POINT(x,y) = 1
3530 ENDPROC
3540 :
3550 DEF PROCexplotar(x_explosion,y_explosion)
3560 REM ** EFECTO SONORO **
3570 SOUND 0,-15,6,50
3580 REM ** ESTABLECER VELOCIDAD FLASH **
3590 *FX9,20
3600 *FX10,50
3610 FOR I = 1 TO 100
3620 MOVE x_explosion,y_explosion
3630 VDU19,2,RND(15),0,0,0
3640 GCOL 0,RND(3)
3650 PLOT 1,RND(100)-50,RND(100)-50
3660 NEXT I
3670 PROCrestaurar
3680 ENDPROC
3690 :
3700 REM ** PROCEDIMIENTOS NIVEL 4 ****
3710 :
3720 DEF PROCconvertir(xcar,ycar)
3730 xgraf = 64*xcar + 32
3740 ygraf = 1023-(32*ycar + 16)
3750 ENDPROC
3760 :
3770 DEF PROCdescubierta_mina
3780 REM ** EFECTO SONORO **
3790 SOUND 2,-15,170,3
3800 REM ** INCREMENTAR MARCADOR **
3810 COLOUR 2
3820 marcador = marcador + 150
3830 marcador$ = STR$(marcador)
3840 marcador$ = LEFT$(cero$,5-LEN(marcador$)) + marcador$
3850 PRINTTAB(11,28);marcador$
3860 ENDPROC
3870 :
3880 DEF PROCrestaurar
3890 contador = contador + 1
3900 IF contador > 4 THEN flag_final = 1: ENDPROC
3910 CLS
3920 VDU19,2,2,0,0,0
3930 COLOUR 2
3940 PROCinicializar_variables
3950 minas_restantes = factor-marcador/150
3960 PROCcolocar_minas(minas_restantes)
3970 PROCtrazar_borde
3980 PRINTTAB(2,27);"Tiempo"
3990 PRINTTAB(2,28)"Marcador"
4000 PRINTTAB(11,28)marcador$
4010 PRINTTAB(2,29)"Max marcador"
4020 PRINTTAB(11,29)max_marcador$
4030 vidas_restantes$ = LEFT$(hombres$,4-contador)
4040 COLOUR 1
4050 PRINTTAB(2,30);vidas_restantes$;"   "
4060 COLOUR 2
4070 PROCsituar_sujetos
4080 ENDPROC
4090 DEF PROCmusica
4100 REM ** PRIMERA BARRA **
4110 SOUND 1,-8,213,5
4120 SOUND 1,-8,209,5
4130 SOUND 1,-8,213,5                                    $
4140 SOUND 1-8,209,5
4150 SOUND 1,-8,213,5
4160 SOUND 1-8,193,5
4170 SOUND 1,-8,205,5
4180 SOUND 1,-8,197,5
4190 REM ** SEGUNDA BARRA **
4200 SOUND1,-8,185,20
4210 SOUND1,-8,165,5
4220 SOUND1,-8,185,5
4230 SOUND1,-8,193,20
4240 REM ** TERCERA BARRA **
4250 SOUND1,-8,165,5
4260 SOUND1,-8,193,5
4270 SOUND1,-8,197,20
4280 ENDPROC
```

Jinetes de la noche

La inteligente combinación de elementos de juegos de guerra y de aventuras con sus avanzadas técnicas de gráficos hacen que este juego marque pautas para el futuro

Los juegos de guerra por ordenador, recreaciones de juegos de tablero como el *Blitzkrieg* y el *Diplomacy*, difieren considerablemente de los juegos de aventuras, la mayoría de los cuales se basan, aunque de forma bastante libre, en el clásico *Dungeons and dragons* (Calabozos y dragones). Un juego de guerra exige la aplicación de estrategia y planes tácticos, con todas las piezas (ejércitos, suministros, armas, etc.) visualizadas en un mapa del campo de batalla. Un juego de aventuras se basa en la sorpresa y el ingenio: un jugador debe resolver una serie de problemas que se le presentan de uno en uno a medida que el juego va avanzando en el escenario. En *Lords of Midnight* (Caballeros de medianoche), para el Spectrum de 48 Kbytes, Beyond Software ha combinado las dos formas para producir una nueva clase de juego.

Al jugador se le proporciona un folleto de 30 páginas que contiene un mapa de la Tierra de Medianoche, en la cual se desarrolla la acción. El jugador representa a Luxor the Moonprince, Lord of the Free (Luxor, príncipe de la Luna y señor de la Libertad). Luxor está en posesión del "anillo de la luna", dispositivo que le permite controlar (y ver a través de los ojos de ellos) a sus cuatro compañeros y a cualquier señor de la Libertad que pueda reclutar. Los señores que la Libertad tienen bajo su control la mayor parte del sur, que se debe defender contra el ataque de Doomdark, the Witchking of Midnight (el rey brujo de medianoche). Las fuerzas de Doomdark las controla el ordenador; con base en una ciudadela situada en el norte, ellos intentan controlar las regiones sureñas.

Doomdark dispone de la ayuda de la "corona de hielo", dispositivo mágico que arroja sobre sus enemigos el "temor glacial". Éste hace disminuir las fuerzas de cualquier señor de la Libertad que se aproxime a las tierras del norte. Sin embargo, uno de sus compañeros, Morkin, es inmune al "temor glacial", de modo que su misión consiste en avanzar sin desmayo hacia el norte para destruir la "corona", mientras los otros personajes se empeñan en luchar abiertamente contra las fuerzas de Doomdark.

Los juegos de guerra tradicionales se basan en un mapa que permite a los jugadores llevar el registro de las fuerzas desplegadas contra ellos. En *Lords of midnight*, los jugadores no disponen de este mapa constantemente actualizado, sino que éste está retenido en la memoria del Spectrum. La panorámica de la acción de que dispone el jugador es la que tiene ante sus ojos un señor de la Libertad, que puede mirar en cualquiera de ocho direcciones que se seleccionan desde el teclado. En consecuencia, si hubiera un ejército enemigo aguardando detrás de la serie de colinas que tiene delante, sería incapaz de verlo a menos que las rodease y mirara en la dirección correcta. Esto confiere a los juegos de guerra una nueva dimensión.

La característica más destacada de *Lords of midnight* son los notables gráficos del juego. Beyond Software afirma que hay 32 000 escenas diferentes y 32 personajes con los cuales se las puede ver. Obviamente, el Spectrum no puede retener tal cantidad de pantallas en su memoria, de modo que Beyond ha desarrollado una técnica llamada "paisajística". Los miles de diversas visualizaciones en pantalla se componen todas de 15 formas diferentes, cada una de las cuales está disponible en cuatro tamaños distintos para dar idea de perspectiva. Estas formas básicas se combinan para posibilitar la construcción de visualizaciones complejas: colinas, bosques, montañas, ejércitos, pueblos y ciudadelas, todo ello ilustrado de forma detallada. La memoria del Spectrum retiene un mapa que proporciona la posición de los elementos en cualquiera de 4 000 ubicaciones. Esta información permite que el ordenador calcule la panorámica desde cualquier punto dado.

Lords of midnight responde a una concepción y una presentación hermosísimas. Para mantenerse a tono con el tema del juego, Beyond incluso ha rediseñado el juego de caracteres del Spectrum para darle cierto sabor gótico a los mensajes del texto.

Lords of midnight: Para el Spectrum de 48 K
Editado por: Beyond Software, Competition House, Farndon Road, Market Harborough, Leicestershire LE19 9NR, Gran Bretaña
Autor: Mike Singleton
Palancas de mando: No se necesitan
Formato: Cassette

Liz Heaney

Ficha indicadora
Lords of midnight viene con una ficha para colocar encima del teclado, mediante la cual se le simplifica al jugador la tarea de buscar las teclas correctas para las instrucciones

Efecto gótico
Según Beyond Software, en el juego hay 32 000 escenas diferentes, ¡aunque dudamos que alguien haya intentado contarlas alguna vez! Observe las letras góticas del texto. Beyond ha rediseñado el juego de caracteres del Spectrum para darle un aire más "teutónico"

Los puntos crecen

Disponemos ya de la rutina del punto, de la línea y del círculo. Nos falta la rutina del relleno (Fillsub), que vamos a analizar en este capítulo y que resulta bastante más compleja de lo que uno pudiera imaginar

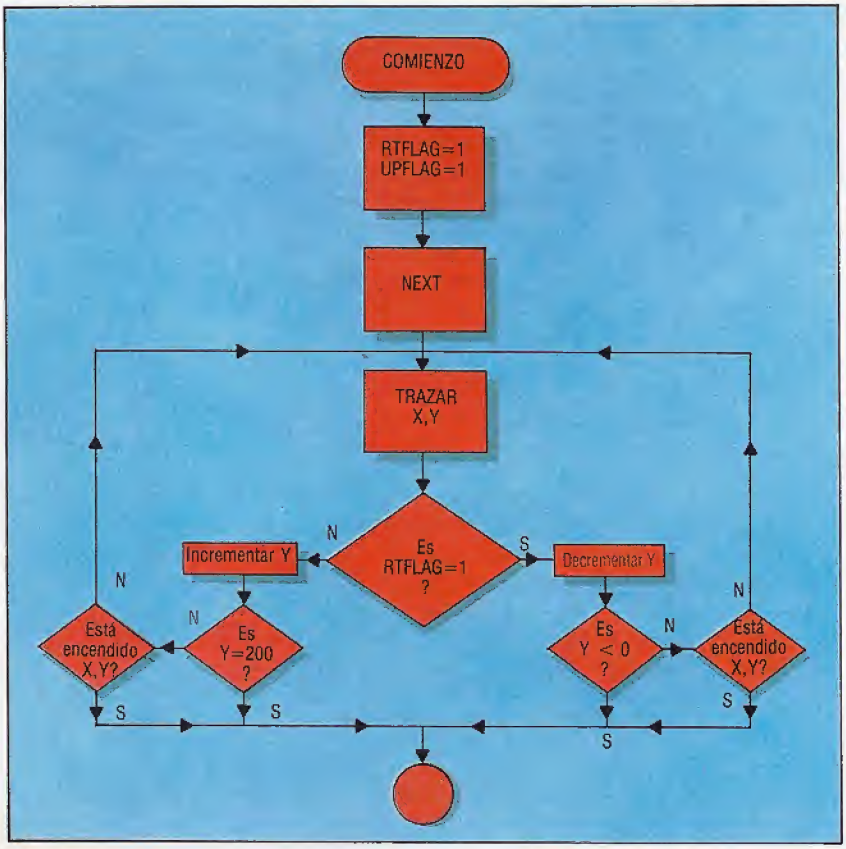
Hay muchos modos de estructurar un algoritmo para rellenar siluetas en la pantalla, pero aunque esto al principio parece tarea fácil resulta más complicado de lo que uno pueda imaginarse. Las figuras que incluyen ángulos "reflejos" (ángulos interiores mayores de 180°) o las que se contraen y se expanden presentan sus dificultades propias. Es posible preparar una rutina que resuelva algunas de estas dificultades, pero no todas. No es cosa fácil dar a un programa la "inteligencia" suficiente para captar lo que constituye una figura cerrada.

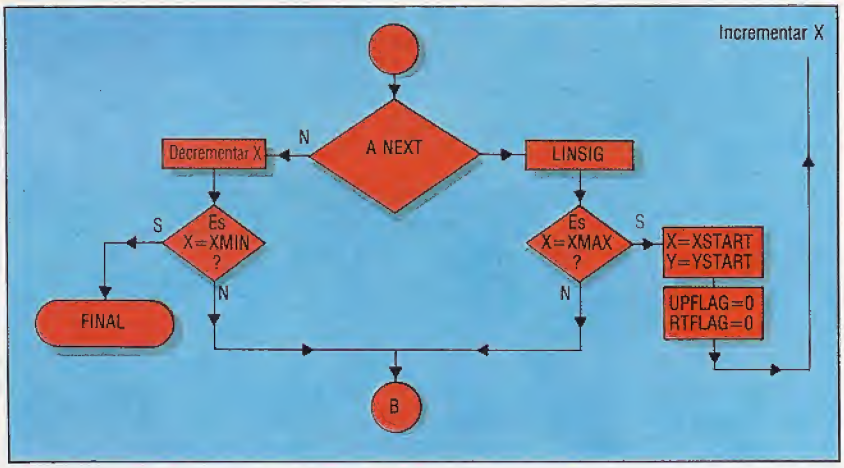
El método que usaremos para nuestra rutina comienza rellenando la figura desde un punto inicial, determinado por el usuario y situado en cualquier lugar dentro del contorno de la figura. La rutina empieza entonces por avanzar pantalla arriba, trazando puntos hasta alcanzar el borde de la figura, a partir del cual se desplaza hacia la derecha un pixel nada más para descender pantalla abajo hasta topar con un nuevo borde. Vuelve a desplazarse entonces un pixel a la derecha para tornar a subir. Este proceso acaba cuando la parte de la figura está rellenada. Pero se vuelve a iniciar en el punto de comienzo moviéndose ahora hacia la izquierda y siempre de arriba abajo y viceversa.

La primera parte de la rutina es inmediata. Dos flags, UPFLAG y RTFLAG (*up*: arriba; *right*: derecha), sirven para indicar la dirección del movimiento de la rutina en cada momento del programa. El primer fragmento del diagrama de flujo muestra la parte de la rutina que controla los incrementos de puntos. El bucle principal incrementa o decrementa el valor de la ordenada Y según el estado de UPFLAG.

Después de preguntar si se trata del borde de la pantalla y comprobar si el pixel siguiente ya está encendido, la rutina cierra el bucle y traza el punto siguiente. Si se encuentra con un pixel ya encendido y con el borde de la pantalla, entonces pasa a la fase siguiente.

La rutina va ahora a moverse a la derecha o a la izquierda según se lo indique el estado de RTFLAG. Es difícil detectar el borde izquierdo o derecho de la figura. La rutina, en vez de hacerlo, deja que el usuario establezca por sí mismo valores máximo y mínimo para la abscisa X. Esto ofrece además la posibilidad de rellenar franjas dentro de la figura.

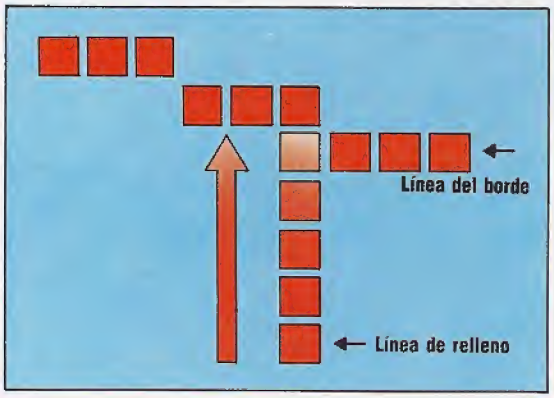




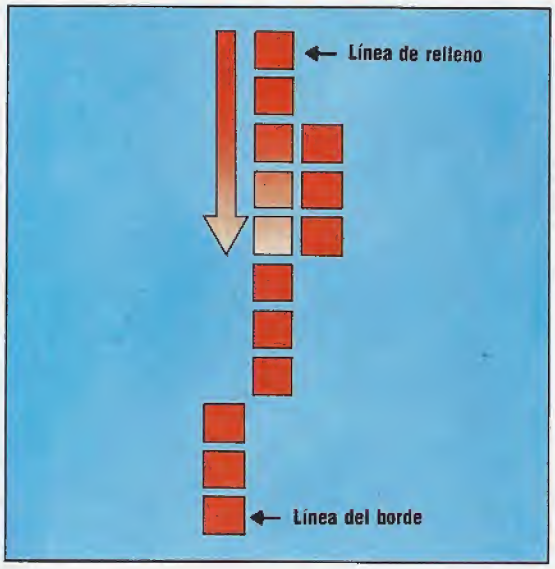
El segundo diagrama de flujo nos muestra cómo los valores de X e Y son restablecidos con los valores iniciales en caso de que, moviéndose a la derecha, alcance el valor máximo de X. Entonces los flags de dirección se ponen a cero como preparativo para rellenar la parte izquierda de la figura. Si el valor de X disminuye hasta alcanzar XMIN, quiere decir que la rutina está acabada. Pero si no se han alcanzado todavía los valores máximos y mínimos de X, la rutina continuará para rellenar la línea siguiente.

Problemas de pendientes

El algoritmo que abarque todo tipo posible de líneas (muy inclinadas o poco, más o menos gruesas) parece que deba ser complicado, pero los principios operativos son relativamente simples. Pensemos en el caso en que la rutina se está moviendo pantalla arriba y encuentra una línea con una pendiente bastante suave.



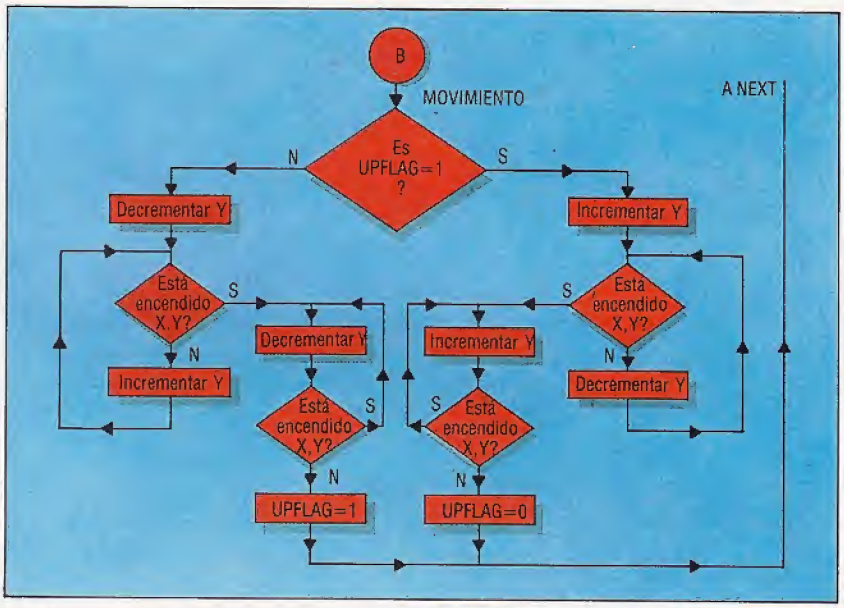
Lo primero que hace el algoritmo al encontrar la línea es retroceder un pixel. Si el movimiento fue a la derecha, éste debe repetirse de modo que pueda iniciarse la siguiente línea en un pixel no iluminado. En el caso de un movimiento descendente de la línea de relleno que encuentra una línea muy inclinada, estaremos ante otro problema por superar.



Si el relleno ha de moverse a la derecha, debe retroceder tres pixels hasta encontrar un espacio vacío desde el cual comenzar su relleno hacia arriba. Pero si ha de moverse a la izquierda descenderá tres pixels más de modo que pueda iniciar su relleno hacia arriba en la primera posición no encendida sobre la línea de borde. Por tanto, para cada dirección de relleno debemos preparar dos bucles para que pueda moverse en una de las dos direcciones hasta la primera posición no encendida. Arriba, a la derecha, diagrama de flujo de este algoritmo.

Listado de la rutina

El cuerpo principal de la rutina Fillsub (*fill*: relleno) sigue bastante de cerca el diagrama de flujo diseñado. Las etiquetas que se usaron en el listado del código fuente se han incluido en las posiciones correspondientes del diagrama para ayudarle a encontrar el recorrido a través del programa.



Kevin Jones

En varios momentos el algoritmo del relleno necesita comprobar si el punto está encendido o no. El programa hace esto con una rutina llamada POINT (punto), que calcula la dirección del punto desde sus coordenadas X,Y de la misma forma que nuestro programa Plotsub. Pero en lugar de realizar un OR lógico para poner el bit a uno, lo que hace POINT es usar AND para ver si el bit en cuestión es uno o no. Si el resultado de la operación AND no es cero, entonces el bit es uno. Este resultado se almacena en PTFLAG para una verificación posterior de parte del programa principal. POINT en gran parte hace el mismo trabajo que Plotsub y los lectores que ya tienen algún dominio del lenguaje del 6502 puede que deseen modificar Plotsub para que encienda un bit en el mapa de bits además de preguntar por el valor de cada bit.

Veamos, por último, cómo se utiliza la rutina Fillsub. Ante todo, Fillsub necesita que se le suministren varios parámetros. Éstos son:

- Las coordenadas del punto inicial. Este punto *debe* estar dentro de la figura que se desea rellenar.
- Los valores máximos y mínimos de la abscisa X.

Teniendo buen cuidado con las figuras con ángulos muy agudos, ya que Fillsub puede moverse fuera

Valioso esfuerzo
Después de estos últimos capítulos y una vez mecanografiados varios kilobytes de lenguaje máquina, llegamos finalmente a conseguir para el Commodore 64 esa facilidad gráfica en alta resolución que cualquier otro microordenador familiar consigue por medio del BASIC. Este esfuerzo le ahorra varios billetes de banco en el caso de que usted no desee utilizar el cartucho de BASIC ampliado

de la figura si llega al borde del espacio interior antes de alcanzar uno de los límites establecidos. El problema se resuelve poniendo los límites un poco más hacia dentro. Observe que la abscisa X del punto inicial y los límites de X deben separarse en la forma byte *hi* / byte *lo*, como se ilustra en el programa de demostración.

Aunque Fillsub no se apoya en ninguna otra rutina ya expuesta para el Commodore 64, las restantes tres rutinas (Plotsub, Linesub y Circsub) también las carga el programa de demostración para dibujar las figuras que ha de rellenar Fillsub.

Cuando esta rutina se creó por primera vez, el relleno se realizaba mediante líneas horizontales y no verticales. Pero se descubrió que el empleo de barras verticales para rellenar figuras se traducía en una ejecución considerablemente más rápida.

Cargador Plotsub/II

Ésta es una versión corregida de la rutina Plotsub que publicamos por primera vez en la página 819. Empléela para crear un nuevo archivo objeto llamado "PLOTSUB.HEX" en cinta o cassette, según se explicó en la citada página

```
10 FORI = 49408TO49408+314
20 READA:POKEI,A:S=S+A: NEXT
30 READCC:IFCC<>STHENPRINT"ERROR SUMA CONTROL"
100 DATA1,0,3,6,0,5,0,0,6,5,5,69,38,2
110 DATA72,138,72,152,72,173,0,193,240
120 DATA83,169,0,133,251,169,4,133,252
130 DATA162,3,160,0,173,2,193,145,251
140 DATA136,208,251,230,252,202,48,8
150 DATA208,244,145,251,160,231,208
160 DATA238,173,1,193,240,24,169,0,133
170 DATA251,169,32,133,252,162,32,160
180 DATA0,169,0,145,251,136,208,251
190 DATA230,252,202,208,246,173,24,208
200 DATA41,240,9,8,141,24,208,173,17
210 DATA208,9,32,141,17,208,76,125,193
220 DATA173,24,208,41,240,9,4,141,24
230 DATA208,173,17,208,41,223,141,17
240 DATA208,104,168,104,170,104,96,72
250 DATA138,72,152,72,173,4,193,141,7
260 DATA193,173,3,193,41,248,141,6,193
270 DATA173,3,193,41,7,141,8,193,173,5
280 DATA193,41,7,141,10,193,162,3,78,5
290 DATA193,202,208,250,173,5,193,141
300 DATA,9,193,169,0,141,11,193,141,12
310 DATA193,162,5,173,11,193,24,109,9
320 DATA193,141,11,193,202,208,243,162
330 DATA6,14,12,193,14,11,193,144,3
340 DATA238,12,193,202,208,242,173,11
350 DATA193,24,109,6,193,141,11,193
360 DATA173,12,193,109,7,193,141,12
370 DATA193,173,11,193,24,105,0,141,11
380 DATA193,173,12,193,105,32,141,12
390 DATA193,173,11,193,24,109,10,193
400 DATA141,11,193,173,12,193,105,0
410 DATA141,12,193,173,11,193,133,251
420 DATA173,12,193,133,252,169,1,141
430 DATA13,193,56,169,7,237,8,193,240
440 DATA7,170,14,13,193,202,208,248
450 DATA160,0,177,251,13,13,193,145
460 DATA251,76,125,193
470 DATA37523:REM"SUMA DE CONTROL"
```

Artificio extraño

La línea 15 DN = 8 indica que los archivos objeto (Plotsub.Hex, etc.) deben cargarse desde un disco. Para usar una cinta cámbiese por DN = 1, y después, haga una cinta con los archivos objeto según el orden especificado en las líneas 20 a 30, o bien, si sus archivos están en cintas diferentes, coloque este código como líneas 22, 26 y 28: INPUT "CAMBIE CINTA Y PULSE RETURN";A$

Demostración Fillsub

```
10 REM *** PROGRAMA DEMO FILLSUB ***
15 DN=8:REM PARA CASSETTE DN=1
20 IFA=0THENA=1:LOAD"PLOTSUB.HEX",DN,1
25 IFA=1THENA=2:LOAD"LINESUB.HEX",DN,1
27 IFA=2THENA=3:LOAD"CIRCSUB.HEX",DN,1
30 IFA=3THENA=4:LOAD"FILLSUB.HEX",DN,1
40 GOSUB1000:REM ESTABLECE ALT RES
50 REM **** DIBUJA TRIANGULO ****
60 XA=100:YA=150:XB=300:YB=160:XC=170:YC=20
80 X1=XA:Y1=YA:X2=XB:Y2=YB:GOSUB2000
90 X1=XC:Y1=YC:GOSUB2000
100 X2=XA:Y2=YA:GOSUB2000
102 REM **** DIBUJA CIRCULO ****
103 XC=60:YC=60:R=50:GOSUB4000
120 REM **** RELLENA TRIANGULO ****
130 XS=170:YS=130:REM COORDENADAS INICIO
140 MIN=100:MAX=299:REM LIMITES
150 GOSUB3000
161 REM **** RELLENA CIRCULO ****
162 XS=60:YS=60:REM COORDENADAS INICIO
163 MIN=10:MAX=109
164 GOSUB3000
```

Demostración Fillsub (cont.)

```
200 GETA$:IFA$=""THEN200:REM ESPERA PULSAR TECLA
210 POKE49408,0:SYS49422:REM RESTABLECE PANTALLA
220 PRINTCHR$(147):REM BORRA PANTALLA
225 PRINT"FIN DE RUTINA"
230 END
1000 REM **** ESTABLECE ALT RES ****
1010 POKE49408,1:POKE49409,1
1020 POKE49410,7
1030 SYS49422
1040 RETURN
2000 REM **** LINESUB ****
2010 MHI=INT(X1/256):MLO=X1-256*MHI
2020 NHI=INT(X2/256):NLO=X2-256*NHI
2030 POKE49920,MLO:POKE49921,MHI
2040 POKE49922,NLO:POKE49923,NHI
2050 POKE49924,Y1:POKE49925,Y2
2060 SYS 49934
2070 RETURN
3000 REM **** FILLSUB ****
3010 SH=INT(XS/256):SL=XS-SH*256
3020 HAX=INT(MAX/256):LAX=MAX-256*HAX
3030 HIN=INT(MIN/256):LIN=MIN-256*HIN
3040 POKE50955,SL:POKE50956,SH
3050 POKE50957,YS
3060 POKE50958,LIN:POKE50959,HIN
3070 POKE50960,LAX:POKE50961,HAX
3080 SYS50967
3090 RETURN
4000 REM **** CIRCSUB ****
4010 CHI=INT(XXC/256):CLO=XC-256*CHI
4020 POKE50497,CLO:POKE50498,CHI
4030 POKE50499,YC:POKE50500,R
4040 SYS 50521
4060 RETURN
```

Cargador de Fillsub

```
10 REM **** CARGADOR EN BASIC DE FILLSUB ****
20 FORI=50944 TO 51375
30 READA:POKEI,A:CC=CC+A:NEXT
40 READA:IFCC<>A THEN PRINT"ERROR SUMA CONTROL":END
100 DATA11,0,6,8,0,3,6,5,136,39,16,60
110 DATA0,60,10,0,109,0,0,1,10,0,16
120 DATA173,11,199,141,20,199,173,12
130 DATA199,141,21,199,172,13,199,169
140 DATA1,141,18,199,141,19,199,140,5
150 DATA193,173,20,199,141,3,193,173
160 DATA21,199,141,4,193,32,131,193
170 DATA173,19,199,208,8,200,192,200
180 DATA240,19,76,82,199,136,192,0,144
190 DATA11,32,246,199,173,22,199,208,3
200 DATA76,46,199,173,18,199,208,31
210 DATA173,20,199,56,233,1,141,20,199
220 DATA173,21,199,233,0,141,21,199
230 DATA205,15,199,208,65,173,20,199
240 DATA205,14,199,208,57,96,173,20
250 DATA199,24,105,1,141,20,199,173,21
260 DATA199,105,0,141,21,199,205,17
270 DATA199,208,34,173,20,199,205,16
280 DATA199,208,26,173,11,199,141,20
290 DATA199,173,12,199,141,21,199,172
300 DATA13,199,169,0,141,19,199,141,18
310 DATA199,76,46,199,173,19,199,208
320 DATA28,136,32,246,199,173,22,199
330 DATA208,4,200,76,191,199,238,19
340 DATA199,136,32,246,199,173,22,199
350 DATA208,247,76,46,199,200,32,246
360 DATA199,173,22,199,208,4,136,76
370 DATA219,199,206,19,199,200,32,246
380 DATA199,173,22,199,208,247,76,46
390 DATA199,72,138,72,152,72,140,2,199
400 DATA173,20,199,141,0,199,173,21
410 DATA199,141,1,199,173,1,199,141,4
420 DATA199,173,0,199,41,248,141,3,199
430 DATA173,0,199,41,7,141,5,199,173,2
440 DATA199,41,7,141,7,199,162,3,78,2
450 DATA199,202,208,250,173,2,199,141
460 DATA6,199,169,0,141,8,199,141,9
470 DATA199,162,5,173,8,199,24,109,6
480 DATA199,141,8,199,202,208,243,162
490 DATA6,14,8,199,46,9,199,202,208
500 DATA247,173,8,199,24,109,3,199,141
510 DATA8,199,173,9,199,109,4,199,141
520 DATA9,199,173,8,199,24,105,0,141,8
530 DATA199,173,9,199,105,32,141,9,199
540 DATA173,8,199,24,109,7,199,133,251
550 DATA173,9,199,105,0,133,252,169,1
560 DATA141,10,199,56,169,7,237,5,199
570 DATA240,7,170,14,10,199,202,208
580 DATA250,160,0,177,251,45,10,199
590 DATA141,22,199,104,168,104,170,104
600 DATA96
610 DATA50785 REM"SUMA CONTROL"
```

Listado assembly

```
                ;++++++++++++++++
                ;++++++++++++++++
                ;++             ++
                ;++    FILLSUB64  ++
                ;++             ++
                ;++++++++++++++++
                ;++++++++++++++++
                ;
                ;++++ VALORES PLOTSUB ++++
                ;
                PLTSUB  =$C163
                XLO     =$C103
                XHI     =$C104
                YLO     =$C105
                MPBLO   =$00
                MPBHI   =$20
                PTR     =$FB
                *       =$C700
                ;
                ;++++ VARIABLES FILLSUB ++++
                ;
                PXLO    *=*+1
                PXHI    *=*+1
                PYLO    *=*+1
                PHBLO   *=*+1
                PHBHI   *=*+1
                PREMX   *=*+1
                PVBYTE  *=*+1
                PREMY   *=*+1
                PROWLO  *=*+1
                PROWHI  *=*+1
                PBPOS   *=*+1
                ;
                XSTLO   *=*+1
                XSTHI   *=*+1
                YST     *=*+1
                XMINLO  *=*+1
                XMINHI  *=*+1
                XMAXLO  *=*+1
                XMAXHI  *=*+1
                RTFLAG  *=*+1
                UPFLAG  *=*+1
                FXLO    *=*+1
                FXHI    *=*+1
                PTFLAG  *=*+1
                ;
                ;++++ PUNTOS DE INICIO A FX,FY ++++
                ;
AD 08 C7                LDA  XSTLO
8D 14 C7                STA  FXLO
AD 0C C7                LDA  XSTHI
8D 15 C7                STA  FXHI
AC 0D C7                LDY  YST
                ;
                ;++++ ESTABLECE FLAGS ++++
                ;
A9 01                   LDA  #$01
8D 12 C7                STA  RTFLAG
8D 13 C7                STA  UPFLAG
                ;
                ;++++ TRAZA PUNTO ++++
                ;
                NEXT
8C 05 C1                STY  YLO
AD 14 C7                LDA  FXLO
8D 03 C1                STA  XLO
AD 15 C7                LDA  FXHI
8D 04 C1                STA  XHI
20 63 C1                JSR  PLTSUB
                ;
                ;++++ INC / DEC COORD Y ++++
                ;
AD 13 C7                LDA  UPFLAG
D0 08                   BNE  DECRY
C8                      INY
C0 C8                   CPY  #$C8           ;TOPE MAX. DE Y?
F0 13                   BEQ  NEXLIN
4C 52 C7                JMP  TESTPT
                DECRY
88                      DEY
C0 00                   CPY  #$00           ;TOPE MIN. DE Y?
90 0B                   BCC  NEXLIN
                TESTPT
20 F6 C7                JSR  POINT          ;PIXEL ILUMINADO
AD 16 C7                LDA  PTFLAG
D0 03                   BNE  NEXLIN         ;SI, BIFURCAR
4C 2E C7                JMP  NEXT
                ;
                ;++++ EMPIEZA SIGUIENTE LINEA ++++
                ;
                NEXLIN
AD 12 C7                LDA  RTFLAG
D0 1F                   BNE  INCRX          ; COMP. PLAG DER.
AD 14 C7                LDA  FXLO
38                      SEC
E9 01                   SBC  #$01
8D 14 C7                STA  FXLO           ;DEC X BYTELO
AD 15 C7                LDA  FXHI
E9 00                   SBC  #$00
8D 15 C7                STA  FXHI           ;DEC X BYTEHI

CD 0F C7                CMP  XMINHI
D0 41                   BNE  MOVE
AD 14 C7                LDA  FXLO           ;X ALCANZA XMIN?
CD 0E C7                CMP  XMINLO
D0 39                   BNE  MOVE
60                      RTS                 ;FIN RUTINA
                ;
                INCRX
AD 14 C                 LDA  FXLO
18                      CLC
69 01                   ADC  #$01           ;INC X BYTELO
8D 14 C7                STA  FXLO
AD 15 C7                LDA  FXHI
69 00                   ADC  #$00           ;INC X BYTEHI
8D 15 C7                STA  FXHI
                ;
CD 11 C7                CMP  XMAXHI
D0 22                   BNE  MOVE
AD 14 C7                LDA  FXLO           ;X ALCANZA XMAX?
CD 10 C7                CMP  XMAXLO
D0 1A                   BNE  MOVE
                ;
AD 0B C7                LDA  XSTLO
8D 14 C7                STA  FXLO
AD 0C C7                LDA  XSTHI          ;RESTABLECE PUNTO INICIAL
8D 15 C7                STA  FXHI
AC 0D C7                LDY  YST
                ;
A9 00                   LDA  #$00
8D 13 C7                STA  UPFLAG
8D 12 C7                STA  RTFLAG         ;RESTABLECE FLAGS
4C 2E C7                JMP  NEXT
                ;
                ;++++ BUSCA COMIENZO SIG. LINEA ++++
                ;
                MOVE
AD 13 C7                LDA  UPFLAG
D0 1C                   BNE  DOWN           ;COMPRUEBA UPFLAG
88                      DEY                 ;Y=Y-1
                AGAIN1
20 F6 C7                JSR  POINT
AD 16 C7                LDA  PTFLAG
D0 04                   BNE  CONT1
C8                      INY
4C BF C7                JMP  AGAIN1
                CONT1
EE 13 C7                INC  UPFLAG         ;PONE A UNO
                AGAIN2
88                      DEY
20 F6 C7                JSR  POINT
AD 16 C7                LDA  PTFLAG
D0 F7                   BNE  AGAIN2
4C 2E C7                JMP  NEXT
                ;
                DOWN
C8                      INY                 ;Y=Y+1
                AGAIN3
20 F6 C7                JSR  POINT
AD 16 C7                LDA  PTFLAG
D0 04                   BNE  CONT2
88                      DEY
4C DB C7                JMP  AGAIN3
                CONT2
CE 13 C7                DEC  UPFLAG         ;PONE A CERO
                AGAIN4
C8                      INY
20 F6 C7                JSR  POINT
AD 16 C7                LDA  PTFLAG
D0 F7                   BNE  AGAIN4
4C 2E C7                JMP  NEXT
                ;++++ FIN PROGRAMA PRINCIPAL ++++
                ;
                ;++++ RUTINA COMPROBACION PUNTO ++++
                ;
                POINT
48                      PHA
8A                      TXA
48                      PHA                 ;REGS. A PILA
98                      TYA
48                      PHA
                ;
8C 02 C7                STY  PXLO
AD 14 C7                LDA  FXLO
8D 00 C7                STA  PXLO           ;TRANSF COORDS
AD 15 C7                LDA  FXHI
8D 01 C7                STA  PXHI
                ;
                ;++++ CALCULA DIRECCION DEL PUNTO ++++
                ;
AD 01 C7                LDA  PXHI
8D 04 C7                STA  PHBHI
AD 00 C7                LDA  PXLO
29 F8                   AND  #$F8
8D 03 C7                STA  PHBLO
AD 00 C7                LDA  PXLO
29 07                   AND  #$07
8D 05 C7                STA  PREMX
                ;
AD 02 C7                LDA  PYLO
29 07                   AND  #$07
8D 07 C7                STA  PREMY
                ;
A2 03                   LDX  #$03
                SHIFT
4E 02 C7                LSR  PYLO
CA                      DEX
D0 FA                   BNE  SHIFT
AD 02 C7                LDA  PYLO
8D 06 C7                STA  PVBYTE
                ;
A9 00                   LDA  #$00
8D 08 C7                STA  PROWLO
8D 09 C7                STA  PROWHI
A2 05                   LDX  #$05
                FIVE
18 08 C7                LDA  PROWLO
18                      CLC
6D 06 C7                ADC  PVBYTE
8D 08 C7                STA  PROWLO
CA                      DEX
D0 F3                   BNE  FIVE
A2 06                   LDX  #$06
                MULT
0E 08 C7                ASL  PROWLO
2E 09 C7                ROL  PROWHI
CA                      DEX
D0 F7                   BNE  MULT
                ;
AD 08 C7                LDA  PROWLO
18                      CLC
6D 03 C7                ADC  PHBLO
8D 08 C7                STA  PROWLO
AD 09 C7                LDA  PROWHI
6D 04 C7                ADC  PHBHI
8D 09 C7                STA  PROWHI
                ;
AD 08 C7                LDA  PROWLO
18                      CLC
69 00                   ADC  #MPBLO
8D 08 C7                STA  PROWLO
AD 09 C7                LDA  PROWHI
69 20                   ADC  #MPBHI
8D 09 C7                STA  PROWHI
                ;
AD 08 C7                LDA  PROWLO
18                      CLC
6D 07 C7                ADC  PREMY
85 FB                   STA  PTR
AD 09 C7                LDA  PROWHI
69 00                   ADC  #$00
85 FC                   STA  PTR+1
                ;
A9 01                   LDA  #$01
8D 0R C7                STA  PBPOS
38                      SEC
A9 07                   LDA  #$07
ED 05 C7                SBC  PREMX
F0 07                   BEQ  BITON
AA                      TAX
                POWER
0E 0A C7                ASL  PBPOS
CA                      DEX
D0 FA                   BNE  POWER
                ;
                ;++++ COMPRUEBA BIT ON ++++
                ;
                BITON
A0 00                   LDY  #$00
B1 FB                   LDA  (PTR),Y        ;CARGA CONTENIDO DIRECCION
2D 0A C7                AND  PBPOS
8D 16 C7                STA  PTFLAG         ;ALMACENA RESULTADO
                ;
68                      PLA
A8                      TAY
68                      PLA                 ;RESTAB. REG. DE PILA
AA                      TAX
68                      PLA
                ;
60                      RTS
```

Calidad garantizada

Softsel es el más importante distribuidor de hardware y software en todo el mundo y cumple una valiosa misión: el control de calidad de los productos

Softsel proporciona a los minoristas de software para ordenador un servicio sumamente valioso. En una época en la que los paquetes de software, algunos de ellos muy caros, están saliendo a la venta cada vez más rápido, un comerciante se encuentra con el problema de evaluar personalmente cada producto nuevo, lo que representa un consumo de tiempo y de dinero, o, de lo contrario, confiar en una evaluación apresurada, lo que podría significar "clavarse" con algunos paquetes invendibles y costosos. El servicio que proporciona Softsel consiste en liberar al minorista de este elemento de riesgo, haciendo una profunda evaluación de producto a todos los paquetes que la empresa incorpora a su lista de software.

Simon Rhodes, director de marketing para Gran Bretaña, explica el procedimiento: "El paquete es examinado primero por nuestro equipo técnico para comprobar su sencillez de uso para el usuario: si está bien programado, bien documentado, si posee buenos gráficos, etc. Luego pasa a nuestro departamento de marketing y ventas, donde se decide si tendrá una buena promoción". Según las estimaciones de Softsel, en un semestre reciente sólo el 10 % de cerca de 700 paquetes se aceptaron para su inclusión en el catálogo de la empresa.

Ian McKinnell

Este control de calidad, junto con los incentivos adicionales de acuerdos de ventas y devoluciones y la necesidad de tratar con un solo proveedor, hacen que una empresa como Softsel sea una propuesta atractiva para un minorista de software. Indudablemente, un minorista podría comprar los paquetes a un precio más reducido directamente de un fabricante, pero la capacidad de Softsel para ofrecer descuentos en función de su potencial de compras en grandes volúmenes hará que la diferencia sólo sea marginal. Simon Rhodes señala: "Aunque un comerciante puede obtener un precio mejor, comprarle directamente a los fabricantes a la larga le costará más, porque tendrá que tratar con cientos de personas diferentes en vez de con una sola empresa".

Orígenes norteamericanos

Softsel fue fundada en 1980 por Robert Leff y David Wagman, quienes habían sido compañeros de trabajo en el departamento de proceso de datos de Transaction Technology, subsidiaria de la gigantesca empresa financiera Citicorp. La creencia de Leff y Wagman de la necesidad de una empresa de las características de Softsel se demostró mediante su rápida expansión. A los cuatro años de su creación, Softsel tiene 350 empleados en todo el mundo y el movimiento total internacional de la empresa durante el último año comercial alcanzó la impresionante cifra de 87 millones de dólares. Sólo en Estados Unidos la empresa posee cuatro enormes almacenes (en Atlanta, Chicago, Los Ángeles y Nueva York), que proveen una gama de 4 500 paquetes a establecimientos minoristas de todo el país.

La penetración de la empresa en el mercado británico comenzó en septiembre de 1982. En abril del año siguiente se creó una subsidiaria, Softsel Computer Products. La filial de Gran Bretaña tiene su base en Feltham, cerca del aeropuerto de Heathrow, y suministra más de 2 500 productos diferentes a minoristas de toda Europa y Oriente Medio.

El futuro de la empresa parece brillante. La subsidiaria británica tiene planeado incrementar el porcentaje de su catálogo de software destinado a paquetes de gestión, que ya representa más de la mitad del catálogo. No obstante, la empresa no tiene intenciones de restarle importancia a la otra principal área de la producción de software, la de los programas de juegos.

Asimismo, Softsel pretende aumentar su participación en el mercado europeo. Ya se ha creado una subsidiaria en Alemania, con oficinas centrales en Munich, y está previsto el establecimiento de subsidiarias en Francia e Italia.

Un paso más allá

En este artículo se pretende arrojar un poco de luz sobre el controvertido ordenador QL de Sinclair

Ian McKinnell

La familia reunida
El nombre Sinclair ha aparecido en una considerable gama de productos: desde amplificadores de audio al celebrado Black Watch, pasando por calculadoras y ordenadores, hasta el tanto tiempo esperado televisor de pantalla plana y el coche eléctrico. Habilidad técnica e innovación, diseño de producto de alta tecnología con estilo y ambiciosas estrategias de marketing han sido los distintivos de Sinclair desde el principio, si bien sus detractores afirmarían que las etiquetas más ajustadas a la realidad serían ingeniería de comunicados de prensa, artimañas y plazos de entrega excesivamente optimistas

No es sorprendente que haya tanta gente confundida sobre el Sinclair QL y su verdadera valía. Primero fue la expectación que despertaron los medios de comunicación a raíz de su lanzamiento. Se le dijo al público que el QL era el ordenador personal más avanzado que existía hasta la fecha, con una CPU 68008 de 32 bits, un juego de cuatro programas de software de gestión integrado y un BASIC residente que superaba a cualquiera de las versiones existentes.

Sin embargo, esta expectación se fue desvaneciendo a medida que las fechas de entrega se retrasaban una y otra vez, y el público comprendió que quizá había ligeras exageraciones en lo que se afirmaba del QL. Se llegó a un auténtico punto muerto cuando quedó en evidencia (a pesar de todas las seguridades dadas por Sinclair) que la máquina no estaba ni por asomo lista para su venta al público: todo el dinero que habían adelantado los clientes mediante pedidos por correo estaba simplemente depositado en las cuentas bancarias de Sinclair produciendo intereses, mientras el ansioso público esperaba y recordaba anteriores lanzamientos de máquinas Sinclair y su entrega.

Entre el lanzamiento del Spectrum y el del QL había, no obstante, una gran diferencia; a pesar de que no se cumplieron los plazos de entrega de 28 días del Spectrum, poco después del lanzamiento se proporcionaron modelos a los periodistas para que los sometieran a examen. Las máquinas funcionaban y eran virtualmente idénticas a las que luego salieron a la venta. En el caso del QL, sin embargo, Sinclair ha tenido que admitir que el "SuperBASIC" y el sistema operativo prometidos no cabían en los 32 Kbytes de ROM asignados: se hubieran necesitado 48 Kbytes, ¡pero en la placa de circuito impreso no había lugar para el chip extra!

En vez de perder tiempo y dinero volviendo a diseñar la placa, Sinclair lanzó su nada afortunado invento denominado *kludge* (llamado algunas veces, equivocadamente, "dongle"). Era una pequeña caja de plástico negro que sobresalía de la puerta para cartuchos del QL y contenía las partes del BASIC y del sistema operativo que no cabían en los 32 K.

Esto al menos le permitió a Sinclair sacar de la fábrica algunas máquinas en condiciones de funcionamiento, con la promesa de que se les quitaría el kludge más adelante, cuando saliera la verdadera máquina mejorada. Los 28 días originales de plazo de entrega se convirtieron en tres meses. Se produjeron en rápida sucesión varias versiones del sistema operativo, cada una de ellas con sus propios defectos. Finalmente, Sinclair se decidió por uno llamado "AH", y éste se ha convertido en la primera versión que sale a la venta en grandes cantidades.

La reacción del consumidor ante el kludge no fue

favorable: era una prueba visible de la insuficiencia de la máquina, y no constitutía ningún indicio de fiabilidad. Para sortear esta reacción, Sinclair tuvo una idea que, con toda seguridad, habrá dejado atónitos a los ingenieros electrónicos: dado que en la placa de circuito impreso no había ningún conector para el tercer chip de 16 Kbytes, éste se colocó "sobre los hombros" del chip existente, y las 28 patas del chip, excepto una, se soldaron individualmente a las del chip de abajo. La última pata se conectó mediante un cable flotante a otra parte del tablero, de modo que se pudiera acceder al nuevo chip independientemente de su anfitrión. Esto supuso la eliminación del kludge, pero en realidad la situación no se había modificado.

Todas las primeras máquinas utilizaban unidades EPROM en vez de ROM, ahorrándole a Sinclair el tiempo que lleva producir unidades de ROM, pero costándole dinero a la empresa. Este alto dispendio posiblemente haya obligado a Sinclair a una temprana aceptación de la versión AH de sistema operativo, a pesar de sus diversos errores. Después de haber arreglado la situación de forma definitiva, se podían fabricar e instalar las ROM para sustituir a las costosas EPROM y la empresa tuvo la esperanza de comenzar a obtener beneficios con la máquina. Lamentablemente, es poco probable que la versión depurada del AH aparezca de inmediato.

Teniendo presente todos estos detalles, ¿cómo podemos juzgar al QL? Se lo puede considerar como dos máquinas en una: un potente ordenador personal o una modesta máquina de oficina y, como tal, se la puede calificar de auténtica pionera. Es probable, sin embargo, que esté más cerca de la idea convencional de lo que es un micro personal: es pequeño, puede producir una visualización en televisión, tiene un BASIC residente, se vende en base a pedidos por correspondencia (y pronto también directamente en tiendas comerciales) y posee gráficos a color en alta resolución y puerta para palanca de mando. Posee dos características que le confieren el estatus de máquina de oficina: los microdrives incorporados proporcionan un sustancial almacenamiento masivo (en comparación con las cassettes) y la máquina viene "arropada" con cuatro programas de aplicaciones: tratamiento de textos, hoja electrónica, base de datos y soporte para gráficos.

Como micro personal el QL parece una muy buena alternativa, dados los microdrives y el software, sobre todo si se considera la alta calidad de éste y el hecho de que los usuarios de ordenadores personales en raras ocasiones ven bases de datos u hojas electrónicas. Por otra parte, la mayoría de ellos no necesitan estas aplicaciones y no tienen ninguna utilización que darles. Al típico usuario de ordenador personal le agrada entretenerse con juegos y escribir programas en BASIC, lo que resulta muy atractivo en este último caso, ya que el SuperBASIC es ciertamente una de las mejores versiones que se han producido hasta la fecha. Existe una escasez obvia de software comercial para el QL, debido a los problemas que se plantearon en la producción de la máquina. La producción de software no se ve facilitada por la incompatibilidad de los nuevos microdrives con la versión para el Spectrum, por el hecho de que la pantalla del QL tenga un trazado distinto, y por las diferencias y los problemas relativos al sistema operativo.

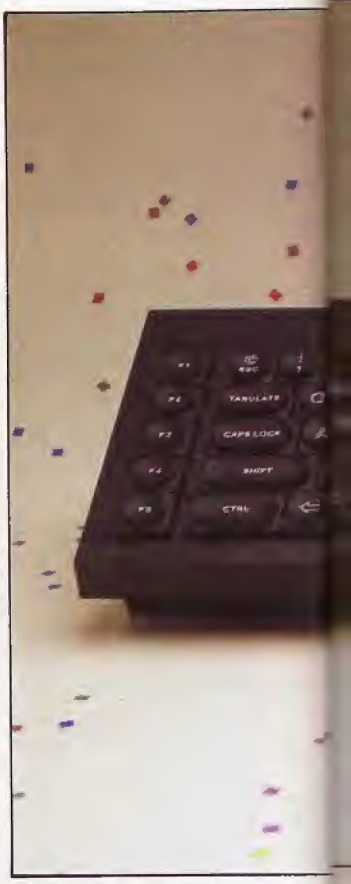
Rivalidad

QL contra BBC Micro
Para aspirar al mercado de usuarios personales serios, el QL debe competir con el BBC Micro. A favor del QL, el BBC Micro no incluye software gratuito, almacenamiento masivo, un microprocesador moderno, gráficos artísticos ni una gran memoria ampliable; por otra parte, el QL no posee la aprobación del gobierno británico para su utilización en las escuelas, ni una enorme base de usuarios en hogares, escuelas y empresas, una inmensa gama de periféricos de alta calidad producidos por terceros, un extenso catálogo de software ni capacidad de ampliación mediante un segundo procesador

QL contra Macintosh
Sir Clive Sinclair fijó la fecha de introducción del QL como para hacerle sombra a la inmitente introducción del Macintosh. Al tomar esta decisión, y al diseñar una máquina basada en el mismo microprocesador, Sinclair ha colocado al QL en una posición de directa rivalidad con el Mac, una máquina de precio cuatro o cinco veces superior. A pesar de que las dos máquinas comparten muchas características, incluyendo el tamaño de memoria, la velocidad del reloj de la CPU y el software integrado, en realidad entre el QL y el Macintosh no cabe ninguna comparación. El QL es una máquina técnicamente brillante, pero no hay nada nuevo en la forma en que opera. El Macintosh de Apple, por otra parte, posee un sistema operativo con iconos, ventanas y el ingenioso "ratón", que coloca a la máquina en un lugar muy alejado de todos los otros ordenadores y la convierte en precursora de una nueva generación de micros

Aparentemente en el SuperBASIC aún persisten algunos errores, y el editor es muy similar al editor de líneas utilizado en el Spectrum; esto no constituye un error, por supuesto, pero tampoco responde exactamente al estándar de la quinta generación. La adición de procedimientos y funciones al estilo BBC y de una estructura SELECT similar a la CASE del PASCAL son mejoras muy reales, si bien no se ofrece ninguna facilidad del tipo de la instrucción ON ERROR. Los gráficos son muy buenos y el sonido, aunque decepcionante, al menos es audible.

Como máquina de oficina el QL es menos convincente: el software que lo acompaña vale la pena, pero seguramente cualquier usuario de gestión de-



Software integrado
Los cuatro paquetes poseen formatos de pantalla similares e instrucciones y visualización claras y concisas. Se pueden trasladar datos entre ellos mediante el microdrive. Los cuatro programas tienen errores en su versión inicial, como instrucciones que no funcionan y errores de entrada-salida; pero éstos resultarán fáciles de solventar en versiones posteriores. Quill, un procesador de textos, permite visualizaciones de 40, 64 u 80 caracteres; los caracteres digitados tardan en aparecer en la pantalla, lo que puede resultar irritante. Abacus es una innovadora hoja electrónica con muchas funciones incorporadas y la capacidad de etiquetar y direccionar un grupo de celdas, pero el espacio de trabajo sobrante, de 15 K, es virtualmente inaprovechable. Archive es una base de datos con instrucciones incorporadas para tareas de archivo sencillas. También se puede programar mediante un lenguaje interno similar al SuperBASIC, pero el microdrive hace que resulte muy lento. Easel crea gráficos de barras, diagramas tipo "tarta" y gráficos lineares de datos numéricos, y se puede pasar rápidamente de un formato a otro

Ian McKinnell

Ian McKinnell

SINCLAIR QL

DIMENSIONES
472 x 138 x 46 mm

CPU
Motorola 68008, 7,5 MHz

MEMORIA
128 K de RAM (ampliables a 640 K); 48 K de ROM

PANTALLA
25 líneas de 80 caracteres (con pantalla); gráficos en alta resolución: 512 x 256 pixels (4 colores), 256 x 256 (8 colores)

INTERFACES
RS232 en serie (2), palancas de mando (2), microdrives, LAN, TV, pantalla RGB

LENGUAJES DISPONIBLES
SuperBASIC

TECLADO
Pseudoestilo máquina de escribir; 65 teclas, incluyendo auténtica barra espaciadora y cinco teclas de función, pero sin tecla de borrado (*delete*)

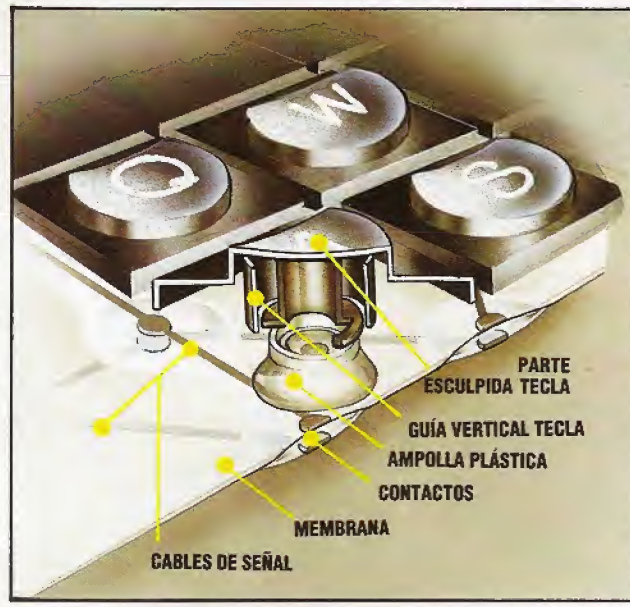
DOCUMENTACION
El manual del usuario responde a un estándar elevado e incluye manuales para SuperBASIC y el software de aplicaciones

VENTAJAS
CPU 68008 de proceso de números muy veloz, software de gran calidad incluido, gráficos muy buenos, versión avanzada de BASIC

DESVENTAJAS
Los microdrives incorporados son más lentos que los discos y no son compatibles con el Spectrum; hay poco software disponible y el sistema operativo no está depurado por completo

seará al menos un paquete de contabilidad, y el programa de hoja electrónica deja sólo 15 Kbytes de RAM para el usuario, lo que descarta a la mayoría de modelos financieros serios. La velocidad y la cuestionable fiabilidad de los microdrives pone en entredicho toda la capacidad de almacenamiento masivo del QL, especialmente al no haber ninguna interface para unidad de disco. El teclado no tiene aspecto de resistir un uso diario intensivo y resulta difícil imaginar que los mecanógrafos cualificados acepten sus peculiaridades. La escasez de software comercial constituye un inconveniente más grave para quienes empleen la máquina con fines de gestión, y esto, unido a las deficiencias del almacenamiento masivo, es probable que dé por concluida la carrera del QL antes de que siquiera comience.

Al igual que todos los productos Sinclair, el QL es atrayente, innovador y objeto de controversia. Aunque en realidad no se puede decir que haya cubierto ninguno de sus objetivos, el QL ha proporcionado a sus competidores un nuevo estándar y un punto de referencia para establecer comparaciones.



Memoria burbuja

El QL posee un teclado tipo "membrana". Dos cables de señal se mantienen separados mediante una burbuja formada en una membrana plástica; cuando se pulsa la tecla, ésta aplasta la burbuja y los cables entran en contacto. La resistencia y la fuerza de retorno las proporciona la "ampolla" plástica. Con teclas plásticas esculpidas de recorrido total, el teclado del QL supone una enorme mejora respecto al del Spectrum, pero su construcción endeble y su tacto esponjoso lo colocan en una situación de desventaja en relación al Vic-20

Rosalind Buckland

Amenaza mortal

El "Ahorcado" es un tradicional juego de palabras cuya implementación para ordenador es sencilla y resulta de gran valor educativo

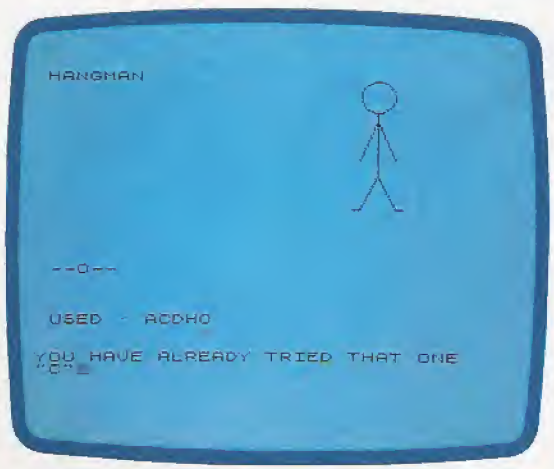
Kevin Jones

Informe de la situación
El juego en plena ejecución, mostrando el desarrollo del hombre colgado y la evolución de la palabra

Probablemente todos hayamos participado en algún momento en el juego del *Ahorcado*. El objetivo del juego consiste simplemente en averiguar las letras de una palabra. La única información que se proporciona es la cantidad de caracteres que posee la palabra, todos los cuales se representan mediante guiones. Una letra adivinada con acierto se visualiza en su posición correcta, y una conjetura incorrecta hace que se dibuje uno de los elementos de la imagen de un hombre sobre un patíbulo. En nuestro programa este dibujo se compone de 10 partes, que se muestran sobre el lado derecho de la pantalla. Si se completan las 10 partes antes de haber completado la palabra entera, entonces el hombre es ahorcado y uno pierde el juego.

El principio básico de nuestro programa es muy simple. Implica verificar si una letra entrada por teclado forma parte de una palabra "secreta" seleccionada al azar (una de once palabras contenidas en sentencias DATA al final del programa). Si la letra aventurada por el jugador está presente, se visualiza en el lugar correcto dentro de la palabra. Si la letra no está presente, el programa debe visualizarla de todas maneras para recordarle al jugador que la misma ya ha sido empleada. Además, el programa debe entonces saltar a una subrutina que dibujará un elemento del hombre colgado.

Las palabras que utiliza nuestro programa están almacenadas, en ambas versiones, en las líneas 1020 y 1030. No existe ninguna razón para no agregar otras, valiéndose de más sentencias DATA. Pero si usted agrega su propio léxico rompedor de cabezas, debe recordar que su longitud no debe superar las diez letras. (Si bien esta limitación se podría ampliar modificando las líneas 30 y 50.) Asimismo, se debe agregar el número total de palabras de las sentencias DATA y alterar consiguientemente el valor N de la línea 20.



Liz Heaney

Al comienzo del programa todas las palabras se leen sobre una matriz. Se escoge al azar uno de los elementos de la matriz y se visualiza en la pantalla una línea de guiones correspondiente a la longitud de la palabra. El resto del juego consiste en un bucle repetitivo. Cuando se entra cualquier cosa por teclado, lo primero es verificarla: si se trata de más de una letra, o de ningún carácter en absoluto, entonces el programa hace un BEEP y vuelve a repetir el bucle en busca de otra entrada. La letra también se verifica con la lista de letras que ya han salido en el juego. Si ya se la ha utilizado antes, entonces aparece en la pantalla un mensaje intermitente y se solicita una nueva letra.

Si la letra aventurada es aceptable, se la agrega a la lista visualizada de caracteres utilizados y luego se la compara con cada una de las letras de la palabra, de una en una. Si concuerda con cualquier posición, se la coloca en la pantalla en lugar del guión correspondiente. Si no concuerda con ninguna letra de la palabra, se llama a una subrutina que dibuja una de las partes del ahorcado.

Cuando el jugador ha hecho 10 conjeturas erróneas, el hombre es ahorcado, se toca una breve melodía de consolación y se selecciona una nueva palabra. De lo contrario, cuando se han colocado correctamente todas las letras de la palabra, se toca una frase musical de felicitación.

Nuestras dos versiones del programa se pueden adaptar fácilmente para la mayoría de los micros personales. Por supuesto, es necesario adaptar las subrutinas que dibujan el hombre y el patíbulo especialmente, de acuerdo a las capacidades para gráficos de cada máquina determinada. Tal vez a los programadores interesados en crear interesantes visualizaciones en pantalla les agrade elaborar más el dibujo final: ¿un hombre ahorcado, colgando suavemente empujado por la brisa cuando el jugador pierde, quizá?

Al juego también se le pueden agregar otros refinamientos. A modo de sugerencia, sería una buena idea agregar una comprobación al principio del programa para ver si todavía no se ha utilizado alguna de las palabras seleccionadas. Tal como está el programa, la selección de éstas es una cuestión de puro azar, y se podría elegir la misma palabra dos veces una inmediatamente después de la otra.

Aún sería más útil una rutina para comprobar las entradas de modo que sólo se aceptaran las letras en mayúsculas. Las letras minúsculas, los números y los símbolos se podrían comprobar mediante sus códigos ASCII. No obstante, tal como lo ofrecemos aquí, nuestro programa es una buena versión del *Ahorcado* y ofrece oportunidades para los programadores audaces.

BBC Micro

```
10 REM                    Ahorcado
20 N=11: REM NUMERO DE PALABRAS EN "DATA"
30 DIM WS(N)
40 REM inicializar
50 DS="_ _ _ _ _ _ _ _ _ _": REM TANTOS GUIONES COMO LONG. MAX.
60 FOR I=1 TO N
70   READ WS
80   WS(I)=WS
90   NEXT I
100 MODE 1
110 REM comenzar una palabra
120 CLS
130 PRINT TAB(5,1);"AHORCADO"
140 RS=WS(RND(N))
150 C=0:W=0:D=0
160 GS=""
170 PRINT TAB(5,23);"Letras probadas :"
180 L=LEN(WS)
190 PRINT TAB(5,20);LEFT$(DS,L)
200 REM entrar la letra aventurada
210 PRINT TAB(5,30);STRING$(34," ")
220 INPUT TAB(5,30);LS
230 REM verificar letra introducida
240 IF LS="" THEN SOUND 1,-15,50,5:GOTO 210
250 IF LEN(LS) > 1 THEN SOUND 1,-15,50,5:GOTO 210
260 F=0
270 FOR I=1 TO LEN(GS)
280   IF LS <> MIDS(GS,I,1) THEN GOTO 330
290   PRINT TAB(5,30)" ESA LETRA YA LA HAS PROBADO"
300   SOUND 1,-15,50,5
310   T%=TIME:REPEAT UNTIL TIME > T% + 50
320   F=1
330   NEXT I
340 IF F=1 THEN 210
350 GS=GS + LS
360 PRINT TAB(5,25);GS
370 REM cotejar letra con palabra
380 F=0
390 FOR I=1 TO L
400   IF LS=MIDS(WS,I,1) THEN GOSUB 480
410   NEXT I
420 IF F <> 1 THEN GOSUB 670
430 IF D=1 THEN GOSUB 590:GOTO 120
440 IF C=L THEN GOSUB 530:GOTO 120
450 GOTO 210
460 END
470 REM letra coincide
480 PRINTTAB(4 + I,20);LS
490 C=C + 1
500 F=1
510 SOUND 1,-15,200,5:SOUND 1,0,0,2
520 RETURN
530 REM exito!
540 FOR I=1 TO 200 STEP 10
550   SOUND 1,-15,I,2
560   NEXT I
570 RETURN
580 REM fracaso!
590 FOR I=200 TO 1 STEP -10
600   SOUND 1,-15,I,2
610   NEXT I
620 CLS
630 PRINT TAB(5,20);"LA PALABRA ERA : ";WS
640 T%=TIME:REPEAT UNTIL TIME > T% + 200
650 RETURN
660 REM la letra no concuerda
670 W=W + 1
680 ON W GOTO 690,760,780,810,830,870,890,910,930,960
690 VDU29,800;800;
700 MOVE 50,0
710 FOR A=0 TO 7 STEP 0.4
720   DRAW 50*COS(A),50*SIN(A)
730   NEXT A
740 VDU29,0;0;
750 RETURN
760 MOVE 800,750: DRAW 800,550
770 RETURN
780 MOVE 740,450:DRAW 750,450:DRAW 800,550
790 DRAW 850,450:DRAW 860,450
800 RETURN
810 MOVE 750,630:DRAW 800,730:DRAW 850,630
820 RETURN
830 PLOT69,800,800
840 PLOT69,820,820
850 PLOT69,780,820
860 RETURN
870 MOVE 780,790:DRAW 800,770:DRAW 820,790
880 RETURN
890 MOVE 600,400:DRAW 1100,400
900 RETURN
910 MOVE 1000,400:DRAW 1000,900
920 RETURN
930 MOVE 1000,900:DRAW 800,900
940 MOVE 1000,850:DRAW 950,900
950 RETURN
960 MOVE 800,900:DRAW 800,850
970 MOVE 780,790:PLOT 14,800,770:PLOT 6,820,790
980 MOVE 780,770:DRAW 800,780:DRAW 820,770
990 D=1
1000 RETURN
1010 REM las palabras
1020 DATA "MUTACION","REEMBOLSAR","SERAFIN","AHORCADO","BBC"
1030 DATA "SPECTRUM","EXODO","ELEFANTE","XENOFOBO","ARRUGA","GRANADA"
```

Spectrum

```
10 REM                    AHORCADO
20 LET N=11
30 DIM XS(N,10)
35 DIM Y(N)
40 REM inicializar
50 LET DS="_ _ _ _ _ _ _ _ _ _"
60 FOR I=1 TO N
70 READ WS
75 LET Y(I)=LEN WS
80 LET XS(I)=WS
90 NEXT I
110 REM empezar una nueva palabra
120 CLS
130 PRINT AT 1,1;"AHORCADO"
140 LET I=(1 + INT (RND*N))
145 LET WS=XS(I)
150 LET C=0:LET W=0:LET D=0
160 LET GS=""
170 PRINT AT 20,1;"UTILIZADAS : "
180 LET L=Y(I)
190 PRINT AT 16,1;DS(1 TO L)
200 REM entrar la letra aventurada
220 INPUT LS
230 REM verificar letra introducida
240 IF LS="" THEN BEEP 0.25,-10:GOT TO 210
250 IF LEN LS > 1 THEN BEEP 0.25,-10:GO TO 210
260 LET F=0
270 FOR I=1 TO LEN GS
280 IF LS <> GS(I) THEN GO TO 330
290 PRINT AT 21,1;"YA HAS PROBADO CON ESA"
300 BEEP 0.25,-10
310 PAUSE 25:
320 LET F=1
330 NEXT I
340 IF F=1 THEN GO TO 210
350 LET GS=GS + LS
360 PRINT AT 20,8;GS
370 REM cotejar letra con palabra
380 LET F=0
390 FOR I=1 TO L
400 IF LS=WS(I) THEN GO SUB 480
410 NEXT I
420 IF F <> 1 THEN GO SUB 670
430 IF D=1 THEN GO SUB 590: GO TO 120
440 IF C=L THEN GO SUB 530: GO TO 120
450 GO TO 210
460 STOP
470 REM letra coincide
480 PRINT AT 16,I;LS
490 LET C=C + 1
500 LET F=1
510 BEEP 0.25, + 16
520 RETURN
530 REM exito!
540 FOR I=-10 TO 10
550 BEEP 0.1,I
560 NEXT I
570 RETURN
580 REM fracaso!
590 FOR I=10 TO -10 STEP -1
600 BEEP 0.1,I
610 NEXT I
620 CLS
630 PRINT AT 10,3;"LA PALABRA ERA : ";WS
640 PAUSE 50
650 RETURN
660 REM la letra no concuerda
670 LET W=W + 1
680 IF W=1 THEN GO TO 690
681 IF W=2 THEN GO TO 760
682 IF W=3 THEN GO TO 780
683 IF W=4 THEN GO TO 810
684 IF W=5 THEN GO TO 830
685 IF W=6 THEN GO TO 890
686 IF W=7 THEN GO TO 910
687 IF W=8 THEN GO TO 930
688 IF W=9 THEN GO TO 960
689 IF W=10 THEN GO TO 960
690 CIRCLE 200,150,10
700 RETURN
760 PLOT 200,140: DRAW 0,-40
770 RETURN
780 DRAW -10,-20: DRAW -4,0
790 PLOT 200,100: DRAW 10,-20: DRAW 4,0
800 RETURN
810 PLOT 190,100: DRAW 10,25: DRAW 10,-25
820 RETURN
830 PLOT 200,150
840 PLOT 196,155
850 PLOT 204,155
860 RETURN
870 PLOT 196,148: DRAW 4,-4: DRAW 4,4
880 RETURN
890 PLOT 170,60: DRAW 80,0
900 RETURN
910 PLOT 240,60: DRAW 0,115
920 RETURN
930 DRAW -40,0
940 PLOT 240,165: DRAW -10,10
950 RETURN
960 PLOT 200,175: DRAW 0,-15
970 OVER 1: PLOT 196,148: DRAW 4,-4: DRAW 4,4: OVER 0
980 PLOT 196,144: DRAW 4,4: DRAW 4,-4
990 LET D=1
1000 RETURN
1010 REM las palabras
1020 DATA "MUTACION","REEMBOLSAR","SERAFIN","AHORCADO","BBC"
1030 DATA "SPECTRUM","EXODO","ELEFANTE","XENOFOBO","ARRUGA","GRANADA"
```

El caso en cuestión

Observe que debe solicitarle al jugador del "Ahorcado" que utilice mayúsculas o bien minúsculas, según sea el caso de la palabra misteriosa

Hacia el Logo

Comenzamos una serie de artículos sobre el LOGO, un lenguaje diseñado con el pensamiento puesto en la educación

Habiendo analizado con todo detalle la programación en BASIC y en lenguaje máquina, iniciamos nuestro curso sobre otros lenguajes de ordenador populares con una serie de capítulos sobre LOGO. Usted se preguntará por qué hemos elegido este lenguaje como tema de un plan de aprendizaje ampliado. Al fin y al cabo, existen muchos otros lenguajes que llevan a cabo ciertas funciones sumamente bien, y son raras las ocasiones en que los ordenadores personales se suministran con LOGO. No obstante, éste ofrece características muy atractivas para el usuario de un ordenador personal.

En primer lugar, el LOGO es uno de los mejores lenguajes introductorios que existen hoy en día para cualquier ordenador. Por supuesto, si usted ya ha estado programando en BASIC quizá no se sienta muy interesado por conocer un lenguaje de introducción. Pero el LOGO puede servir, incluso para un programador de BASIC ya experimentado, como una excelente introducción a la programación "estructurada" y para el empleo de procedimientos en lugar de sentencias. En segundo lugar, el LOGO está disponible en cartucho o en cassette para la mayoría de los ordenadores personales. De hecho, una de las mejores versiones de LOGO existentes es la escrita para el Spectrum que distribuye Sinclair. Por último, el LOGO es un sistema de aprendizaje muy eficaz. Aunque no es sencillo de dominar, el LOGO es uno de los pocos lenguajes de programación con el que es fácil empezar a trabajar.

El LOGO tiene sus orígenes en el lenguaje de inteligencia artificial LISP, que se desarrolló a principios de los años sesenta para que los ordenadores pudieran tratar con más facilidad estructuras de datos complejas. Su nombre proviene del hecho de que es un lenguaje de "proceso de listas", lo que significa que su estructura de datos básica es una lista, en vez de una serie de caracteres o de una matriz numérica, como en el BASIC. Las funciones esenciales del LISP manipulan los datos dentro de una lista. Los elementos de la lista pueden ser símbolos simples o listas enteras. La ventaja de este enfoque es que de esta manera los datos no numéricos (como una oración) se procesan más fácilmente.

El LISP depende en gran medida del principio de la recursión, donde algo (por lo general una función o procedimiento) se define en términos de sí mismo. En el caso del LISP, el ítem que se define siempre es una lista. Éstas no son características accidentales del LISP, sino que surgen a raíz de sus orígenes en las investigaciones basadas en ordenadores sobre el lenguaje natural y la inteligencia artificial. Sin embargo, no se trata de un lenguaje fácil de aprender, y en 1968 un grupo de personas relacionadas con el Massachusetts Institute of Technology (MIT) se propuso desarrollar un lenguaje para niños basado en el LISP.

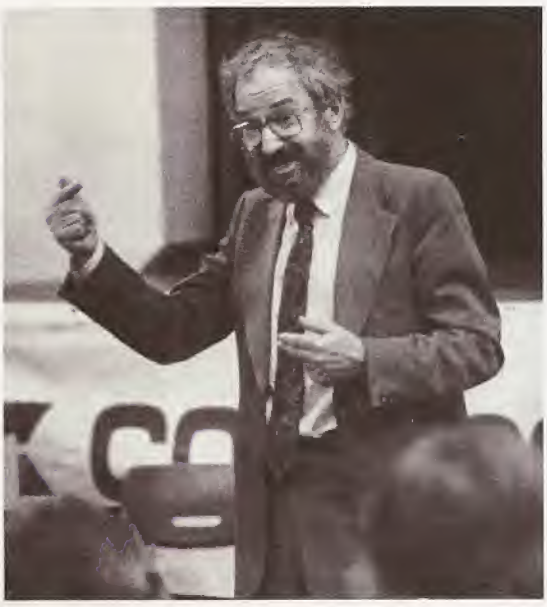
El líder del grupo del MIT era el carismático Seymour Papert. Anteriormente había estado estudiando durante algunos años el desarrollo cognosci-

Presentando el LOGO

El LOGO posee dos características fundamentales que hacen del mismo un lenguaje educativo tan eficaz. La primera es que es interactivo: cuando se digita una instrucción, inmediatamente se ven los resultados en la pantalla. Ello significa que es fácil ir avanzando (en especial para los niños y los principiantes) porque el alumno puede ir controlándose a sí mismo paso a paso.

La segunda característica esencial es que el LOGO es ampliable: operaciones completas se manipulan mediante listas de instrucciones de LOGO elementales. Estas listas se denominan *procedimientos*. Una vez que se ha definido un procedimiento como un conjunto de instrucciones determinadas, el nombre de ese procedimiento asume el estatus de una nueva instrucción de LOGO. A partir de entonces, el procedimiento entero se puede ejecutar simplemente digitando su nombre.

Al programar en LOGO muchas personas tienden a ser más exploradoras que en otros lenguajes. Algunas veces seguirán un enfoque bastante estricto y definirán un esquema específico desde el mismo comienzo. Otras veces empezarán con un problema central y escribirán un procedimiento para resolverlo, y luego construirán un programa alrededor de ese procedimiento. Se puede asumir un enfoque flexible al LOGO porque suelen existir varios caminos para llegar a un resultado determinado.



Creador y teórico
Seymour Papert, creador del LOGO, aparece en la fotografía durante un congreso que patrocinó Commodore en 1983. En la actualidad Papert está asociado con LOGO Computer Systems, Inc., (LCSI), que suministra programas de LOGO para el Sinclair Spectrum, los ordenadores Atari y otras máquinas

tivo (aprendizaje) en los niños pequeños con Jean Piaget (1896-1980), el eminente psicólogo educacional de su generación. Al pasar al MIT, Papert comenzó a trabajar en estrecha relación con un experto en inteligencia artificial, Marvin Minsky. En su trabajo con el LOGO Papert intentó unir las ideas de sus colegas, unificando teorías de aprendizaje cognoscitivo e inteligencia artificial.

El trabajo sobre el LOGO siguió durante los años setenta y se formaron otros grupos para experimentar con el nuevo lenguaje. El más notable de ellos fue el de Edimburgo. Todo su trabajo de desarrollo se llevó a cabo en departamentos de investigación universitarios utilizando ordenadores centrales o miniordenadores. El advenimiento de los micros supuso la difusión del LOGO a niveles más amplios.

El LOGO es un lenguaje sofisticado que requiere muchísima memoria, tanto para las instrucciones como para espacio de trabajo. Los intérpretes de LOGO de los micros típicamente exigen alrededor de 30 Kbytes de memoria, y otros ocho Kbytes o más para la visualización de gráficos. ¡Todo esto antes de que uno empiece siquiera a programar! En consecuencia, a pesar de que fue posible implementar intérpretes de BASIC simples en micros personales desde el mismo momento en que éstos se comercializaron, el LOGO para micros sólo se convirtió en una posibilidad factible cuando se pusieron al alcance del gran público los ordenadores personales de más de 48 Kbytes de RAM.

Pero fue *Mindstorms* (Basic Books, 1980), de Seymour Papert, la que sacó al LOGO de los departamentos de investigación y lo convirtió en foco de atención de un grupo mucho mayor de personas. En este libro, Papert desarrolla una visión de la forma en que se podrían utilizar los ordenadores en la educación. Este trabajo es el resultado de la síntesis de tres grupos de ideas: teorías del desarrollo cognoscitivo, inteligencia artificial y el movimiento en la educación orientado hacia el aprendizaje centrado en el niño. Lo que Papert desea es ver a los niños programando ordenadores, en vez de a los ordenadores programando al niño.

El libro postula la aparición de una nueva "cultura del ordenador", en la cual las ideas "formales" que previamente se consideraban fuera de las capacidades del niño podrían ser manipuladas por éste con facilidad. El niño sería capaz de hacerlo debido a la forma en la que habría utilizado los ordenadores para explorar ideas formales. Es esta exploración de ideas, activa, cooperativa y no estructurada (alumno-alumno y alumno-maestro), lo que constituye la "filosofía LOGO" sobre la que se basa la utilización del lenguaje en el campo de la educación.

Papert escribe y convence por el puro poder de su retórica. Sin embargo, en la teoría existen algunos problemas graves. Hay pocas pruebas experimentales que la sustenten, a pesar de algunos estudios realizados; las teorías de Piaget acerca del desarrollo cognoscitivo se convierten en una prescripción para la educación en forma que él jamás pretendió; y existen áreas de solución de problemas (¡incluso en matemáticas!) que el LOGO no cubre.

A medida que los maestros van utilizando más ampliamente el LOGO en la clase, están descubriendo que no todo funciona de la forma en que Papert lo describe y no están obteniendo los resultados que habían esperado. Existe el peligro del desencanto, pero una vez dejado de lado el entusiasmo excesivamente desproporcionado acerca de lo que el lenguaje puede hacer, el LOGO aún sigue siendo una excelente forma de introducir conceptos informáticos, de explorar ciertas clases de ideas y de desarrollar aptitudes para la resolución de problemas.

Versiones autorizadas
En la fotografía vemos las versiones más amplias y mejor documentadas del LOGO para el Commodore 64 (Terrapin-MIT), Sinclair Spectrum y ordenadores Atari (LCSI). Aunque caras, éstas son las versiones de LOGO que han sido autorizadas por los fabricantes y que se asemejarán en mayor grado al lenguaje MIT original. El LOGO para el Commodore 64 se vende en disco; el LOGO Atari viene en cartucho, y el LOGO Sinclair es una versión basada en cassette

En los micros actuales el LOGO tiene muy poco espacio de trabajo y opera con excesiva lentitud. Hasta cierto punto es un lenguaje que está a la espera de que el hardware alcance sus niveles de exigencia. Pero como lenguaje para aprendizaje no tiene ningún competidor serio.

¿Para quién es el LOGO?

¿A quién le puede ser útil aprender a programar en LOGO? Nosotros creemos que muchas personas, incluso los programadores experimentados, pueden aprender muchísimo a través de la programación en LOGO, incluyendo:

- Cualquiera que sea un recién iniciado en informática o en programación;
- Cualquiera a quien agrade jugar con los ordenadores y piense que su uso debe ser divertido;
- Quien se vea frustrado por una falta de poder expresivo en otro lenguaje de programación;
- Cualquiera que esté interesado en el pensamiento, el aprendizaje o la enseñanza;
- Quienes deseen una visión de áreas más avanzadas de la informática, especialmente quienes se dedican al estudio de la inteligencia artificial.

Habiendo dicho esto, debemos recordarle que, al igual que la programación en BASIC y en lenguaje máquina, el LOGO no es para todos. Específicamente, el LOGO podría no ser el mejor lenguaje para:

- Quien crea que utilizar ordenadores es "trabajar". Algunos lenguajes están diseñados para el trabajo, así como los caballos de tiro. Pero un caballo de tiro es lo último que uno escogería para una cabalgata por el campo;
- Quien desee o espere muchísima velocidad del ordenador para el proceso de sus instrucciones. El LOGO utiliza muchísima memoria y opera lentamente en la actual generación de micros. (En programas comparables, el LOGO puede funcionar a la mitad de velocidad que el BASIC.)

Incluso para estos grupos de personas puede ser valioso cierto conocimiento del LOGO: se puede usar para bosquejar una solución a un problema y prepararla para su traducción a otro lenguaje.

Pronto llegará...

la Tortuga

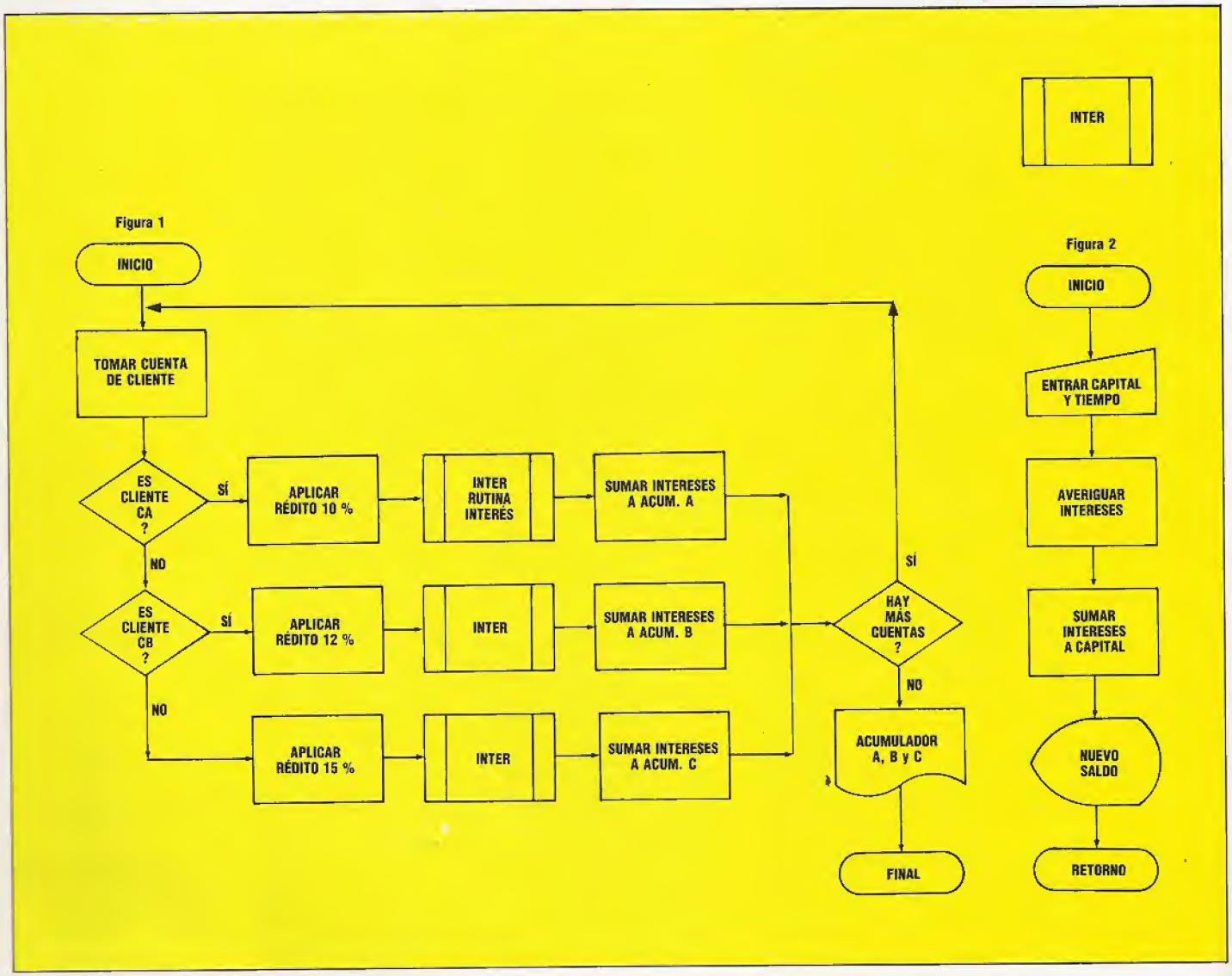
Uso de rutinas

Ahora emplearemos una rutina en nuestro último ejemplo, para optimizar su solución

La solución dada al problema aparecido en el capítulo anterior mostraba cómo llegar al resultado deseado, pero a base de la repetición de una misma parte de la secuencia para cada uno de los tipos de cliente. Con anterioridad nos hemos referido a la utilidad de los bucles o bien a la agrupación en un solo punto de las partes repetitivas, pero en este caso en particular pudo comprobarse que tal alternativa no era viable (véase p. 968, fig. 2). A continuación se muestra cómo optimizar la solución mediante el empleo de una rutina.

Una rutina podría definirse como aquella parte del programa que, debido a que se repite en diferentes ocasiones a lo largo del mismo, se separa del cuerpo principal, indicándose su uso mediante un símbolo especial (véase arriba, a la izquierda).

Funciona de la siguiente forma. Supongamos que tomamos la cuenta de un cliente de la segunda categoría. Tras una respuesta negativa en la primera decisión, adoptará la vía correspondiente a cliente del tipo CB. Luego de darle valor al rédito (12), que es la única operación que no puede ser incluida en la rutina, el símbolo siguiente hace referencia al uso de la misma. Ésta recibe el nombre de INTER y está representada aparte. La rutina tiene su propio inicio, que es la conexión del programa principal con ella. También tiene su final, que en este caso no es físico, sino que retorna el control al programa justo en la instrucción siguiente a su llamada. Así, todas las veces que se pase por tal símbolo, el proceso continuará en la rutina, volviendo a su origen al final de ésta.



Nuevas expresiones

Los avances en la música electrónica son asombrosos: analizaremos aquí la codificación digital del sonido, o muestreo

A pesar de los muchos desarrollos que se han producido en este campo durante los últimos sesenta años, vale la pena recordar las características simples de voltaje controlado de un oscilador. Cuando un objeto físico cualquiera (ya sea el ala de una abeja, o una cuerda vocal humana) vibra, el aire circundante se expande y se contrae muy rápidamente, produciendo una forma de onda que el oído y el cerebro humano interpretan como sonido. Si a una diminuta lengüeta de metal se le aplica un voltaje eléctrico a través de un modulador (como una bobina de inducción de automóvil), el metal vibra, creando la forma de onda más simple: la onda sinusoide. La altura, o frecuencia de vibración, de la oscilación resultante depende del voltaje aplicado y, aunque en menor grado, de la densidad de la tira de metal. Esta diminuta unidad de generación de sonido se denomina *oscilador*. El control de voltaje ha sido el principal método para producir música sintetizada durante décadas.

La interface MIDI, que se anunció por primera vez en 1983, es una unidad que está diseñada para permitir que un sistema digital (como un ordenador) controle a otro, como, por ejemplo, un sintetizador. Su desarrollo es consecuencia de los adelantos que se vienen sucediendo en la producción de música electrónica en la última década.

Durante varios años los estudios de grabación han incorporado muchas piezas diferentes de equipo de proceso de sonido; con frecuencia se considera como prueba concluyente de la valía de un estudio una impresionante gama de unidades de filtro y reverberación. Del mismo modo, en las actuaciones en vivo de un músico de los años setenta que se sirviera de un sintetizador, éste debía estar enteramente rodeado por bancos de teclados, cada uno de ellos con multitud de controles.

Al considerar lo que sucede en un estudio de grabación bien equipado, es útil pensar en el juego de sociedad "Susurros chinos". En este juego se va pasando una frase de persona a persona. El último jugador recita luego lo que él ha escuchado de la oración original. Así, una frase sencilla y directa puede haberse convertido en un conjunto de palabras sin sentido o con éste totalmente distorsionado. El proceso que tiene lugar en el estudio de grabación es similar, pero en este caso la frase original es un conjunto de sonidos musicales. Y en lugar de una cadena de oyentes, cada uno de los cuales transmite una versión tergiversada del original, aquí hay un grupo de unidades para proceso de sonido, cada una de ellas controlable y con una tarea específica a cumplir. Cualquiera que se sirva de este equipo probablemente utilizará un tablero de conexiones centrales para hacer las conexiones o, de lo contrario, conectará las unidades entre sí directamente. Un ingeniero de sonido experimentado puede calibrar los controles de cada unidad de forma manual en cuestión de segundos, pero es fácil imaginar los problemas de sincronización y comunicación inherentes a esta clase de conexiones.

Trío innovador
Alannah Currey, Tom Bailey y Joe Leeway (de izquierda a derecha), de los Thompson Twins. En sus comienzos un grupo *arty* de siete componentes, actualmente actúan como trío con el apoyo de ritmos secuenciados grabados en cinta. (Se suele denominar *arty* a aquel conjunto musical cuyos integrantes en su mayoría provienen de escuelas de arte o conservatorios.)

El músico "de teclado" de los años setenta tenía un problema diferente. En su caso, la dificultad inmediata no era cómo producir un sonido con cada instrumento en sucesión: el músico sólo tenía que mover la mano de un teclado a otro. Lo más probable era que se le presentaran dificultades cuando los dos teclados se tenían que tocar al mismo tiempo, pero los músicos de clavicordios y de los órganos de las iglesias supieron cómo hacerlo durante siglos. Incluso música tan compleja como una fuga de Bach, se podía ejecutar en un solo teclado y, de todos modos, la mayor cantidad del "trabajo" de estudio no se hacía en una sola "sesión". En cambio, cada parte del sintetizador se grababa sola, superponiendo las partes subsiguientes sobre la primera, utilizando distintos sectores de una cinta de canales múltiples. Pero ejecutar este tipo de música requería un artista diestro, o dos músicos corrientes, o bucles de cinta y un ingeniero de sonido.

Pero el desarrollo más importante de los años setenta fue el de presentar las unidades generadoras de sonido *dentro* del sintetizador, y las técni-

Centro de control
La interface Roland MP401 MIDI es una sofisticada unidad que conecta sintetizadores digitales a microordenadores. Controla todas las funciones de sincronización externa, el tiempo interno y externo, la entrada/salida de cinta y la salida del sintetizador. Ello significa que el ordenador anfitrión sólo es responsable del envío y la recepción de instrucciones y de la gestión de la memoria. La MP401 funciona con el IBM PC y el Apple IIe. Según Roland, en un futuro cercano habrá una versión disponible para el Commodore 64

cas instrumentales en general. Un buen ejemplo de esto es el nacimiento del sintetizador de bajos controlado por teclado. En los años sesenta, el estilo Motown de música pop-soul dependía en gran medida de los bajos eléctricos. Durante los años setenta los bajistas *funk* alcanzaron un nivel de virtuosismo que rivalizaba con el de los principales guitarristas; a finales de la década este estilo ya no pudo evolucionar más y surgió el sintetizador de bajos controlado por teclado. Esto supuso nuevos problemas al músico de teclado, a quien ahora se le exigía que asumiera las funciones del bajista (trabajando con el tambor para "sujetar" la sección de ritmo en un tiempo estricto) y que ahora no estaba tocando música de "teclado". A medida que aparecieron en el mercado nuevos sintetizadores analógicos se dispuso de sonidos de trompeta, saxofón y tambor. Más y más músicos de teclado se volcaron a un dispositivo simple para asumir todas estas nuevas responsabilidades. Éste era el *secuenciador*.

Un secuenciador es un dispositivo que coordina varios ramales de música independientes unificándolos en una pieza única según un patrón especificado. Opera mediante la utilización de voltajes controlados que se generan a partir de un oscilador para producir una serie de tonos de frecuencias diferentes. Cuanto mayor es el voltaje, más rápido vibra la tira metálica y la forma de onda resultante se escucha como un sonido agudo. Para controlar el oscilador se utiliza una unidad de secuenciación. Ello es necesario porque la música muy raramente se compone de frases ininterrumpidas; se requieren breves vacíos o largos silencios, tanto para crear patrones rítmicos como para conformar la estructura global de la música. Un vacío en un patrón de frases se produce cuando la unidad de control le envía al oscilador un voltaje cero. La secuenciación tiene por objeto asegurar que el vacío se produzca en el mismo lugar cada vez que se repite el patrón.

A fines de los años setenta eran pocos los sintetizadores que tenían facilidades completas de secuenciación, pero los músicos fueron rápidos para utilizar aquello de lo que disponían. La vibrante música disco que produce Giorgio Moroder con Donna Summer es en gran parte música de secuenciador, y los nuevos grupos británicos de sintetizador desarrollaron un estilo completamente nuevo para ponerlo a la altura de los nuevos equipos. Ya no es necesario tocar cada nota individual con una serie de gestos grandilocuentes, al estilo de Rick Wakeman. En cambio, toda una secuencia, o *riff* (término que podríamos definir como "conjunto de notas definitorias o características de una canción"), se podía iniciar o interrumpir modificando un control. En el ínterin el músico se podía alejar

Steve Cross

Contar y tocar

Se pueden producir sonidos electrónicos programando la salida de tono y/o generadores de ruido blanco (así es como uno crea sonido en un microordenador), o también mediante el muestreo de sonidos reales, haciendo un "modelo" digital de los sonidos y utilizando ese modelo para recrear la forma de onda de los sonidos a través de un convertidor de digital a analógico que active un altavoz. El muestreo proporciona una imagen de sonido exacta (según la cantidad de muestras de forma de onda almacenadas), que de lo contrario sería difícil de programar excepto mediante ensayo y error: ¡intente sintetizar en su micro el ritmo de un tambor pequeño!

Analógico
La salida del micrófono es simplemente una forma de onda de tiempo de voltaje: una analogía eléctrica del sonido. La aplicación de esta forma de onda a un altavoz reproduce el sonido

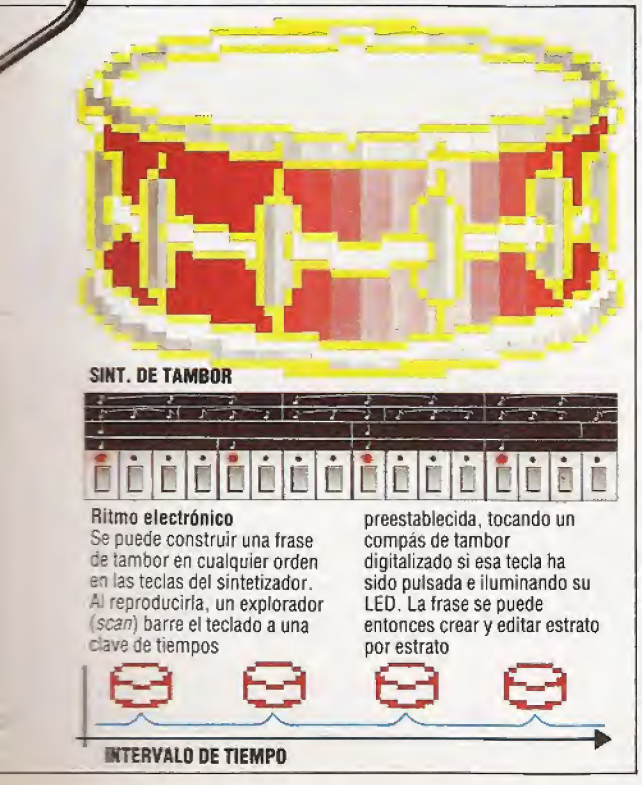
Digital
El convertidor de analógico a digital "digitaliza" la forma de onda continua en una secuencia de mediciones de voltaje discretas. Cuantas más mediciones de éstas se tomen, más exacta será la imagen digital del sonido

del teclado, bailar al ritmo del compás del secuenciador y después regresar a otro teclado para tocar una melodía o una serie de acordes. A mediados de los ochenta, todo el estilo de actuación de un grupo como los Thompson Twins está determinado por la existencia del secuenciador.

Cuando aparecieron por primera vez los sintetizadores digitales, a menudo su diseño se modelaba en función de sus predecesores analógicos. Los músicos descubrieron que los nuevos instrumentos desarrollaban aún más sus "aptitudes de secuenciación", y es en esta área del control digital donde se ha generado el mayor interés. Ello lo demuestra la reciente popularidad de la batería eléctrica Linn, una de las primeras unidades en utilizar el sonido muestreado, en este caso proporcionado por Steve Gadd, el mejor batería norteamericano. En la máquina Linn, los patrones de tambor se graban en forma digital del mismo modo en que los datos del ordenador se graban en discos flexibles. Las secuencias resultantes de unos y ceros se codifican entonces en chips de ROM. Mediante el acceso a un chip determinado, un músico o un productor puede reproducir el sonido original como si se lo estuviera tocando en vivo. La gran ventaja de digitalizar el sonido es que la salida puede ser alterada en la consola, apartándose del patrón original en cuanto a tiempo, ritmo, volumen, etc.

Ahora nuestros dos ejemplos originales (el estudio de grabación y el músico con sintetizador en el escenario) tienen ciertas características en común. Además de grabar música, el trabajo en el estudio refleja el *boom* del video que se ha producido en estos últimos años. El video ha aportado un nuevo estilo y una nueva conciencia en la fabricación de imágenes: los productores exigen que la música que lo complementa refleje este hecho.

Si la música que acompaña a un video se produce mediante instrumentos convencionales, la sincronización con la imagen en pantalla es muy similar a la



Ritmo electrónico
Se puede construir una frase de tambor en cualquier orden en las teclas del sintetizador. Al reproducirla, un explorador (*scan*) barre el teclado a una clave de tiempos preestablecida, tocando un compás de tambor digitalizado si esa tecla ha sido pulsada e iluminando su LED. La frase se puede entonces crear y editar estrato por estrato

Cortesía de Record Mirror

En vanguardia
Donna Summer, con el productor Giorgio Moroder, fue una de las primeras artistas pop que utilizó en la música grabada sintetizadores y ritmos producidos electrónicamente

que se realiza con una película. Sin embargo, si la música se compone de sonidos individuales y frases producidas por sintetizadores digitales, hay muchas cosas que pueden ir mal. Imaginemos que en un determinado video una vasija con flores se cae al suelo, donde se hace añicos. El músico que trabaja en la partitura ha creado una frase que se acelera hasta el punto en el que la vasija cae, y ha dispuesto un acorde de percusión para el momento en que se estrella contra el suelo. Comienza la grabación y empieza la frase, pero en seguida resulta evidente que la velocidad de la aceleración se ha calculado mal y la frase concluye mientras la vasija todavía está sobre la mesa. El músico prueba entonces el acorde de percusión, que se graba en un canal distinto de la cinta. La grabación se reanuda, y en esta ocasión el acorde se graba una fracción de segundo más tarde. Los músicos necesitan una forma de conjuntar los instrumentos digitales de modo que todo suceda en el momento adecuado. Lo que se precisa, entonces, es una interface digital.

El músico de sintetizador tiene problemas similare, esta vez en una actuación en directo. Su equipo incluye dos sintetizadores digitales, producidos por diferentes fabricantes, y una batería eléctrica o caja de ritmos Linn. El músico tiene preparadas frases en un sintetizador y en el Linn, pero como éstos no llevan juntos el compás, suele acabar por dejar que el Linn funcione automáticamente mientras toca el otro de forma manual. El resultado es que su segundo sintetizador, adquirido en razón de la calidad de sus sonidos previamente preparados, queda sin intervenir. Este músico necesita una forma de unir sus instrumentos de modo que el secuenciado de todo el material se produzca en el punto correcto. Además es indispensable que el secuenciador de su primer sintetizador toque los sonidos preestablecidos del segundo. Por otra parte, sea cual fuere el equipo que utilice, debería ser aplicable a algo más que a sus propios sintetizadores: ¡bien podría nuestro músico descubrir su verdadera identidad en el estudio a que hemos aludido anteriormente, que planteaba tantos problemas en la sincronización de video!

El factor humano

En el diseño de programas es esencial la interface hombre-máquina, que se ocupa del intercambio de información entre usuario y programa

Durante muchos años la programación de ordenadores ha sido un tema misterioso comprendido sólo por profesionales que podían dedicar mucho tiempo y esfuerzo a la materia. Antes del advenimiento del microordenador con su teclado tipo máquina de escribir, con frecuencia los programas eran entrados byte a byte a través de interruptores situados en el panel frontal del ordenador, o perforando agujeros en cintas en la consola de un teletipo.

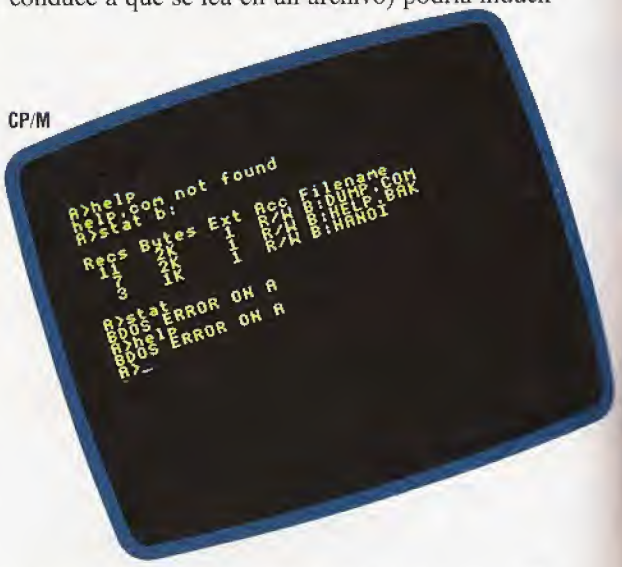
En comparación, el usuario es actualmente una criatura mimada. Los fabricantes ya no esperan que el propietario de un ordenador luche con el lenguaje máquina, y se ha acuñado la frase "amabilidad hacia el usuario" para indicar que cualquiera, independientemente de su experiencia, puede utilizar y programar micros. En 1982 el Alvey Committee, en el informe *A programme for advanced information technology* (Programa para la tecnología avanzada de la información), identificaba la interface hombre-máquina (MMI: *man-machine interface*) como una de las cuatro áreas principales de investigación y desarrollo, junto con la ingeniería de software, el diseño de circuitos integrados a muy gran escala (VLSI: *very large scale integration*) y los sistemas basados en el conocimiento.

En cualquier aplicación la interacción entre el ordenador y el usuario, por la cual se pasan entre uno y otro datos o instrucciones, reviste una importancia capital. Este "diálogo" se conduce a través de los dispositivos de entrada/salida (E/S) del ordenador, con el teclado actuando como principal fuente de entrada y la pantalla de visualización proporcionando la salida. Palancas de mando, lápices ópticos, "ratones", pantallas al tacto y otros dispositivos también se pueden utilizar para la entrada, mientras que el ordenador puede servirse de una impresora, un generador de sonido (o de voz) o incluso un robot para manifestar la salida.

Además de cualquiera de las limitaciones que pueden plantear los dispositivos de E/S utilizados, el diálogo entre usuario y máquina se ve influido por el software. Por ejemplo, el sistema operativo (OS) del ordenador controla muchos detalles de la operación del teclado y la pantalla. La velocidad a la cual se repiten las teclas cuando se mantienen pulsadas y la demora entre repeticiones las establece el sistema operativo, que también se guarda las pulsaciones de teclas para permitir al ordenador almacenar caracteres que se han digitado más rápidamente de lo que los mismos se pueden visualizar. Esto es importante porque tiene relación directa con la velocidad a la cual el usuario puede introducir información. El tamaño del buffer es vital y el usuario debería conocerlo; el sistema operativo CP/M, por ejemplo, guarda en el buffer una única pulsación de tecla, mientras que muchas máquinas personales almacenan 10 pulsaciones o más.

Pero los buffers de tecla pueden plantear problemas. Un usuario experimentado que esté trabajando con un sistema activado por menú puede saber de antemano que las opciones de menú que él requiere son la 2 del menú principal, la 5 del menú siguiente, luego la 3, 4, 6, etc. Dado que él está familiarizado con el sistema, entra sus opciones a gran velocidad. Con un buffer de 10 caracteres, el usuario terminará allí donde quiera, porque todas las pulsaciones de teclas se "recordarán" en la secuencia correcta. Con un buffer de un solo carácter, el tiempo necesario para visualizar el segundo menú puede que sea mayor que el tiempo que se precise para digitar la secuencia entera de opciones. Por consiguiente, en vez de seleccionar la opción número 5 de este menú, después la 3 y así sucesivamente, sólo se realiza la opción número 6 (porque éste es el único carácter retenido en el buffer) y el sistema se detiene allí.

Pero un buffer grande también puede suponer algunos problemas. Un programa de menú que demore mucho tiempo para reaccionar ante una pulsación de tecla (lo que puede ocurrir si la opción conduce a que se lea en un archivo) podría inducir



al usuario a pensar que no está pasando nada. La respuesta natural en este caso sería probar la última opción introducida, pulsando luego una selección de teclas hasta que hubiera una respuesta. Esto podría llevar a que el programa intentara procesar los caracteres ilegítimos retenidos en el buffer; ¡los resultados podrían ser sorprendentes!

La "recogida de basura", que implica limpiar los registros de memoria del ordenador para dejar espacio de trabajo libre, constituye otra fuente de problemas. Esta operación puede hacer que un programa parezca "colgado" durante largos perío-

dos, en los cuales el usuario puede nuevamente tratar de emprender una acción correctiva. Es probable que la recogida de basura produzca problemas en programas largos que realicen mucha manipulación de series. Algunas versiones de BASIC permiten que el programador provoque una recogida de basura; es una buena idea hacerlo a intervalos frecuentes, pero es conveniente transmitir al usuario un mensaje "espere por favor", dado que parecerá que el ordenador no esté haciendo nada mientras se ocupa de la recogida de la basura.

La forma en que un lenguaje de programación manipule la entrada y la salida influirá en el diseño de la interface entre el ordenador y el usuario. Las superiores facilidades para manipulación de series del BASIC conducirán a un ejemplo más sofisticado de series en el diálogo hombre-ordenador que el que permiten lenguajes como el PASCAL. Los BASIC que posean instrucciones incorporadas para posicionamiento del cursor favorecerán trazados de pantalla mejores que aquellos que no las posean. Lo mismo puede decirse de los BASIC con instrucciones para gráficos. El BASIC está bien provisto de instrucciones de entrada/salida; INPUT y PRINT están bien para programas sencillos. Pero para un verdadero control de entrada, intente experimentar con

"Log-on" (Conexión) al Micronet 800

GET$, INKEY$, INPUT$() e instrucciones similares. PRINT USING es una instrucción sumamente versátil para formatear la salida; tiene un incalculable valor para alinear puntos decimales y para justificar columnas de texto.

En cualquier sistema hombre-máquina, el elemento más impredecible es el usuario. Sin embargo, a diferencia de cualquier otro componente, el usuario posee ciertas características de rendimiento que se han de comprender antes de diseñar la interface.

Las personas comparten con los ordenadores la característica básica de ser "procesadores de información". No obstante, los seres humanos tienen limitaciones inherentes respecto a la cantidad de información nueva que pueden retener en la "memoria de trabajo"; se ha comprobado que para la mayoría de ítems de información se pueden retener simultáneamente en el cerebro alrededor de siete ítems diferentes. El tamaño de estos ítems depende de lo significativos que sean o de lo bien estructurados que estén. Si la información a recordar se compone de caracteres al azar, cada ítem se compondrá de no más de un único carácter. Pero si los caracteres no son al azar sino que son apellidos comunes, cada ítem recordado podría ser un nombre completo. Al incrementar la estructura de la información de esta manera, se incrementa la capacidad del usuario para recordarla y utilizarla.

Existen varias formas de ayudar a la gente a estructurar la información cuando utilizan ordenadores. Un método consiste en relacionar los datos con estructuras familiares y bien comprendidas; ésta es la forma en la que funciona la visualización del "escritorio" estilo Lisa. Del mismo modo, se podría organizar un paquete de hoja electrónica financiera como para que tuviera el aspecto de un libro, con páginas, índices, etc. Otro procedimiento consiste en entrenar al usuario para que comprenda estruc-

Apple Macintosh

Ian McKinnell

turas que no le sean familiares. Mostrando ejemplos repetidamente y explicando temas en profundidad, se podría utilizar el propio programa para enseñarle al usuario cómo se debería estructurar la información. El entrenamiento de esta clase tiene el inconveniente de que resulta caro, tanto en tiempo como en esfuerzo. Instrucciones detalladas, pantallas de "ayuda" y "postes indicadores" pueden proporcionar un tipo de entrenamiento en línea, pero son difíciles de utilizar con eficacia.

Por último, presentar la información en patrones reconocibles puede ayudar al usuario a comprender el programa. Esto se puede hacer valiéndose del color o del trazado para guiar el ojo hacia la información deseada. Para entender lo que esto significa, consideremos la codificación con colores como la que utilizan el Prestel y programas de videotexto similares. En una página típica, el encabezamiento y el pie estarán dispuestos en bloques del mismo color; habrá un único color de fondo y el texto se visualizará en otros dos colores, alternando un párrafo en cada color. Las palabras clave se pueden realzar aplicando aún otro color. Todo ello tiene por finalidad permitir que el usuario seleccione sólo la información requerida y que ignore secciones enteras de la página si éstas contienen información que carece de un valor inmediato. Sin embargo, la codificación con colores puede ser fuente de confusión si se la emplea con exceso.

Tres niveles
Estas fotografías de sistemas operativos de microordenadores ilustran tres niveles variables de amabilidad hacia el usuario. En la primera fotografía (de izquierda a derecha) un usuario nuevo está intentando comunicarse con el sistema operativo CP/M. El CP/M no posee facilidades de "ayuda" incorporadas, de modo que para que se lo pueda utilizar adecuadamente es necesario un profundo conocimiento de las instrucciones. Nuestro segundo ejemplo es un sistema conducido por menú: el menú "Log-on" para el Micronet 800 en el BBC Micro. Las opciones están claramente numeradas y el usuario realiza su elección entrando el número apropiado que se le indica en el menú. La pantalla no ofrece gran cantidad de información, de modo que el usuario debe comprender las opciones para poder hacer uso de ellas. Nuestra tercera fotografía muestra el sistema operativo del Apple Macintosh, que proporciona indicaciones visuales y visualizaciones gráficas, así como menús sencillos y fácilmente comprensibles

Abierta al mundo

Cómo acceder a la puerta para el usuario con el fin de controlar fenómenos físicos externos al ordenador

Muchos de los micros personales poseen puertas para el usuario que permiten la entrada al mapa de memoria del ordenador en virtud de una serie de conexiones eléctricas. La base de todos los sistemas digitales es que el sistema binario de unos y ceros se pueda representar mediante dos niveles de voltaje. Normalmente, un cero se representa mediante 0 voltios y un uno mediante +5 voltios.

Cada posición de memoria se compone de un grupo de ocho celdas individuales, teniendo cada una de ellas un nivel de voltaje de 0 o 5 voltios. El patrón de estos niveles de voltaje determina, por consiguiente, el número que está almacenado en esa posición de memoria. Si una celda posee un nivel de voltaje de +5 voltios, decimos que está *activada (set high)*, y si la celda posee un nivel de 0 voltios, se dice que está *desactivada (set low)*. Las conexiones externas de la puerta para el usuario están unidas eléctricamente a una o más posiciones de la memoria del micro y mediante la lectura de los valores de estas posiciones, o mediante la escritura de valores en ellas, podemos monitorizar o controlar sistemas eléctricos fuera del ordenador.

Existen dos tipos de conexiones para puerta para el usuario. Algunas puertas tienen grupos de patillas separados (ocho para entrada y ocho para salida) conectadas con dos posiciones de memoria. Otros utilizan las mismas ocho patillas tanto para entrada como para salida. Aquí vamos a analizar el segundo tipo de configuración, que es el que emplean los micros BBC y Commodore 64.

Registros de dirección de datos

Además de tener una posición conectada con las ocho patillas de la puerta para el usuario, los micros con puertas bidireccionales hacen uso de una segunda posición de memoria, conocida como el *registro de dirección de datos* (DDR: *data direction register*). Este registro determina si cada una de las ocho líneas ha de enviar o recibir datos. Un uno en el DDR coloca a una línea en modalidad de salida, y un cero la coloca en modalidad de entrada. Para situar las ocho líneas de la puerta para el usuario en

Enchufando

Comenzamos nuestro proyecto haciendo cables para el BBC Micro y el Commodore 64. Estos cables se utilizarán para conectar las máquinas con el exterior a través de un formato común de ocho líneas de datos con una línea a tierra a cada lado. Se necesitará:

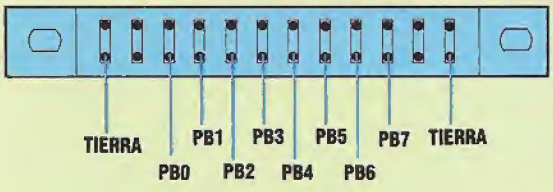
BBC Micro
- enchufe IDC de 20 vías
- cable plano IDC de 20 vías (un metro)
- estaño para soldar y soldador

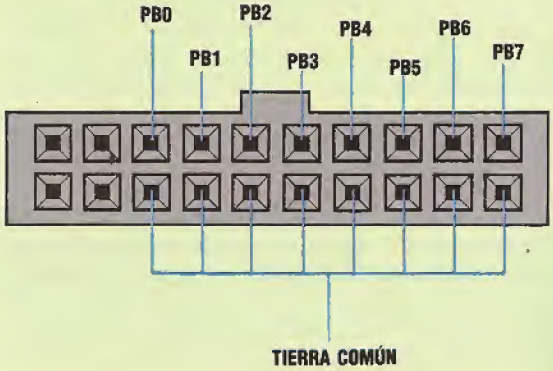
Commodore 64
- conector marginal de 0,15″ y 24 vías
- cable plano de 10 vías (alrededor de un metro)
- estaño para soldar y soldador

Los diagramas de las patillas de la puerta que vienen en los manuales de las máquinas muestran la posición de las 10 líneas (dos a tierra, ocho de datos) que necesitamos. El enchufe IDC del BBC posee una patilla de posicionamiento a uno de los lados que se separa en dos partes desiguales: sostenga el enchufe verticalmente con la patilla lejos de usted y su sección de ajuste arriba, y pase por la misma más o menos una pulgada de cable plano con la franja roja a su derecha. Cierre el enchufe sobre el cable apretándolo firmemente (tal vez con una prensa de tornillo o tornillo de ajuste). Separe y recorte 10 líneas por el otro extremo según la ilustración, luego pele y estañe los extremos de las líneas restantes (véase p. 524).

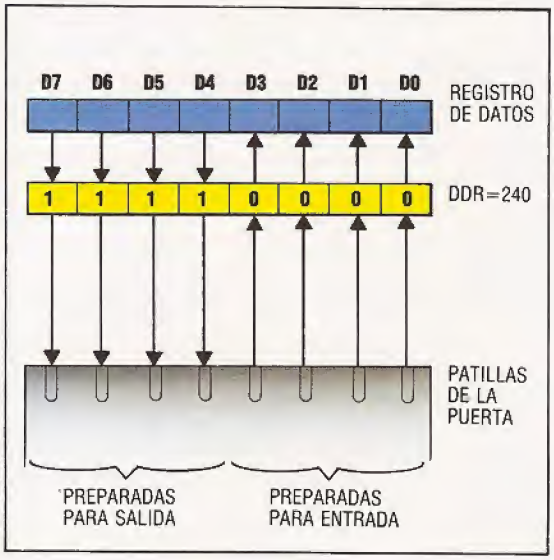
Para el Commodore 64, marque uno de los lados del enchufe claramente como la parte superior (y al conectarlo a la máquina tenga siempre este lado hacia arriba). Pele y estañe los dos extremos de las 10 líneas y suelde el cable a las patillas inferiores del conector marginal.

Utilice los programas de prueba y un tester (véase p. 567) para comprobar sus cables





salida, el DDR habrá de establecerse en 255 (es decir, 11111111 en binario). Del mismo modo, las ocho líneas se pueden establecer para aceptar entrada estableciendo el valor del DDR en cero. Las ocho líneas se pueden configurar en cualquier combinación de líneas de entrada o salida estableciendo en el DDR el valor apropiado. Por ejemplo, las cuatro líneas más significativas de la puerta para el usuario se podrían establecer en modalidad de salida, y las cuatro menos significativas en entrada, colocando en el DDR el valor 240 (es decir, 11110000 en binario).



Los registros de datos y de dirección de datos poseen las siguientes direcciones:

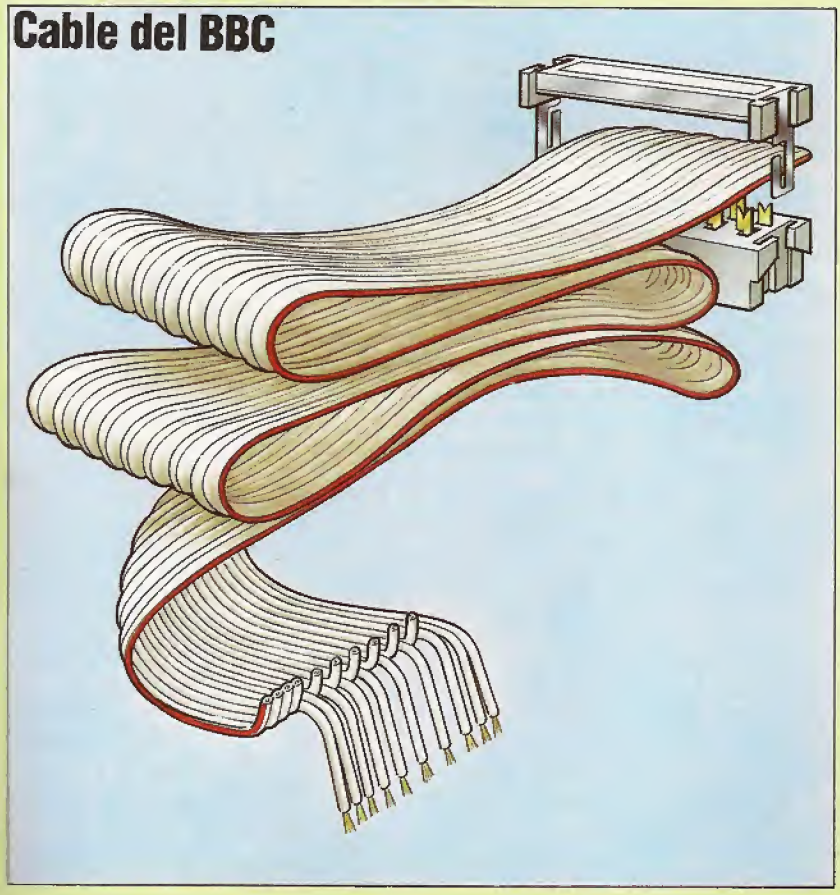
Tipo de micro	Registro de datos	Reg. de direc. de datos
BBC Micro	&FE60 (65120 dec.)	&F362 (65122 dec.)
Commodore 64	$DD01 (56577 dec.)	&DD03 (56579 dec.)

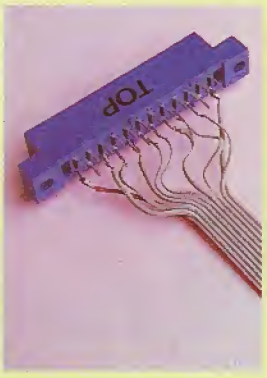
El siguiente programa coloca la puerta para el usuario de manera que se puedan usar las ocho líneas para entrada, y visualiza el registro de datos:

```
4 REM ********** C64 **********
5 REM *  VISUALIZACION REGDAT  *
6 REM ********** C64 **********
10 DIM A$(10):A$(0)="E":A$(1)="H"
20 REGDAT=56577:DDR=56579
30 POKE DDR,0:REM=ENTRADA SOLO
50 :
100 PE=PEEK(REGDAT):GOSUB 500
150 PRINT "REGDAT=";PE;"=";B$
200 GOTO 100
300 :
499 REM ****************************
500 REM *  S/R CONVERSION BINARIO  *
501 REM ****************************
550 B$="":N=PE
600 FOR D=1 TO 8
650 N1=INT(N/2):R=N-2*N1
700 B$=A$(R) + B$:N=N1
750 NEXT D:RETURN
```

Para el BBC, introduzca estas modificaciones:

```
20 REGDAT=&FE60:DDR=&FE62
30 ?DDR=0
100 PE=?(PE):GOSUB 500
```

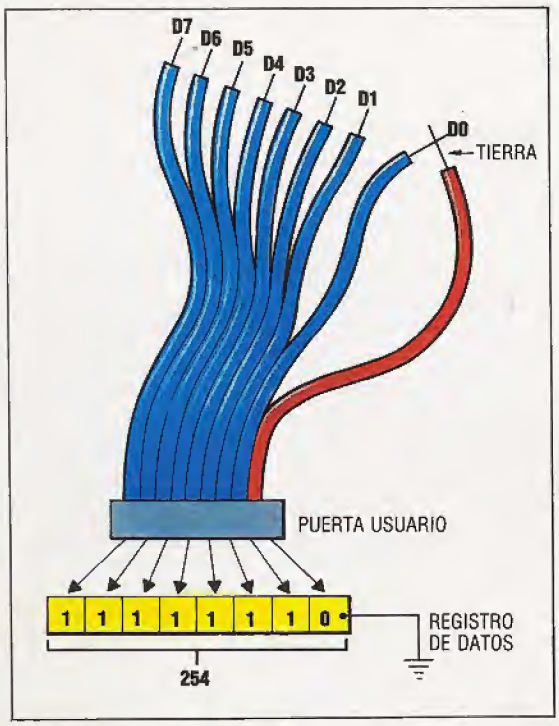




Líneas de conexión
Los cables de la puerta para el usuario del BBC Micro y el Commodore 64

La ejecución del programa muestra que el contenido normal del registro de datos es 255 (en la pantalla HHHHHHHH), lo que significa que las ocho líneas están activadas y las ocho celdas de REGDAT están en el nivel de voltaje +5. Si ahora enchufa un cable en la puerta para el usuario, puede utilizarlo para cambiar los niveles de voltaje en las líneas y, por lo tanto, cambiar el contenido numérico de REGDAT.

Hemos cableado 10 líneas a la puerta para el usuario: ocho de ellas son líneas de datos (una línea para cada bit de REGDAT) y las otras proporcionan una "toma de tierra" al sistema. Hacer que una línea de datos toque una línea de tierra reducirá el nivel del voltaje de la línea de datos a cero, cambiando por consiguiente los niveles de voltaje de REGDAT. Si se conecta a tierra D0, por ejemplo, mientras se está ejecutando el programa, verá que ahora REGDAT contiene 254 (reflejado en la pantalla como HHHHHHHE), lo que significa que la línea menos significativa está en voltaje *tierra* (0 v), mientras que las otras están *altas* (+5 v). Quizá quiera probar distintas combinaciones de líneas para comprobar que, mediante este procedimiento, puede hacer que REGDAT contenga cualquier número entre cero y 255.



Escribiendo software

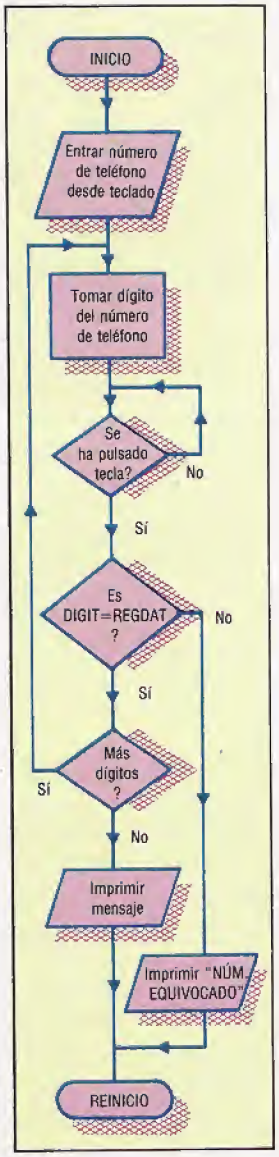
La esencia de la toma de decisiones del ordenador es la comprobación de valores numéricos o condiciones y la bifurcación según el resultado obtenido en la comparación. Por lo tanto, es una cuestión sencilla diseñar software que pueda monitorizar actividades físicas a través de la puerta para el usuario. Mediante la comprobación del valor del registro de datos y emprendiendo la acción apropiada, el ordenador puede responder a los cambios externos. Un ejemplo sencillo sería el de diseñar un programa que compruebe si un número de teléfono se ha marcado correctamente. El "marcado" se puede efectuar conectando a tierra ciertas líneas de datos conectadas a la puerta para el usuario, con el fin de producir en el registro de datos el valor numérico correcto para cada uno de los dígitos del número de teléfono. A causa de las dificultades que entraña conectar a tierra simultáneamente varias líneas de datos, el programa espera que el usuario pulse una tecla del teclado antes de analizar el contenido del registro de datos. La lógica del programa se refleja en el diagrama de flujo que lo acompaña.

```
100 REM ************** C64 ***************
101 REM*          NUMEROS DE TELEFONO          *
102 REM ************** C64 ***************
120 :
130 REGDAT=56577:DDR=56579
140 POKE DDR,0:REM=ENTRADA SOLAMENTE
150 :
160 INPUT"NUMERO DE TELEFONO";NT$
170 :
180 FOR K=1 TO LEN(NT$)
190 DG$=MID$(NT$,K,1): REM TOMAR DIGITO
200 IF DG$ <> " "THEN GOSUB 500
210 NEXT K
250 :
300 PRINT"  --------  LO SIENTO----------  "
310 PRINT" NO HAY NADIE EN EL ";NT$
320 PRINT" POR FAVOR LLAME MAS TARDE"
350 PRINT:PRINT:RUN
400 :
500 REM** S/R CONVERSION Y COMPROBACION **
550 DG=VAL(DG$)
600 PRINT"PONER DIGITO EN LAS LINEAS"
650 PRINT"      Y PULSAR CUALQUIER TECLA"
700 GET GT$:IF GT$="" THEN 700
750 PE=PEEK(REGDAT):IF PE=DG THEN PRINT
    DG" OK":RETURN
800 :
850 PRINT"???NUMERO EQUIVOCADO???"
900 PRINT NT$"<>"LEFT$(NT$,K-1);PE
950 PRINT"???VUELVA A PROBAR???"
999 PRINT:PRINT:RUN
```

Para el BBC, introduzca estas modificaciones:

```
130 REGDAT=&FE60:DDR=&FE62
140 ?DDR=0
700 A=GET
750 PE=?(REGDAT):IF PE=DG THEN
    PRINT DG" OK":RETURN
```

Hemos examinado aquí la entrada de información desde una fuente externa con el fin de afectar el flujo de control dentro de programas. En el próximo capítulo analizaremos la salida a través de la puerta para el usuario y el diseño de un sistema sencillo de control por ordenador.



Líneas cruzadas
El programa de números de teléfono acepta como entrada un número de teléfono (cualquier formato, cualquier longitud) y verifica sus dígitos, uno por uno, con el contenido de la posición de memoria de la puerta para el usuario. Los espacios en el número introducido se ignoran. El programa aguarda una pulsación de tecla antes de comparar un dígito introducido con el dígito de la puerta para el usuario

Ejercicios de programas

Basándose en el programa de números de teléfono:

1) Escriba un programa que simule la acción de una alarma antirrobo.

2) Escriba un programa para contar el número de impulsos recibidos por la puerta para el usuario en un período dado.

3) Modifique su última respuesta para llevar una cuenta separada para cada una de las ocho líneas de datos de la puerta para el usuario.

4) Escriba un programa que simule la acción de una cerradura con combinación.

5) Escriba un programa para cambiar el color de la pantalla desde la puerta para el usuario.

Magia negra

"Necromancer" (Nigromante) es un juego con espectaculares gráficos y en tres actos, como si se tratara de una obra de teatro

Introducción. Tanto la versión Atari como la del Commodore 64 se cargan fácilmente; el lento arrastre de la indolente unidad de disco del Commodore 64 resulta tolerable debido a que el programa produce una gama de extraños ruidos que suenan algo así como una orquesta electrónica afinando. Los gráficos de los títulos se acompañan con una música de presentación que aprovecha al máximo las posibilidades de sonido de ambas máquinas.

Primer acto. El bosque: La pieza comienza en la "edad de la oscuridad", cuando el "supremo soberano es Tetragorn, el malvado hechicero". El jugador hace el papel de Illuminar, un druida al que se describe como "el defensor de la verdad y protector de la raza humana". Como puede imaginar, ello no es tarea fácil. Comienza la obra y aparece el druida en un espacio abierto oscuro. Una atmósfera de estrellas lo protege de centenares de pequeños ogros que marchan sin descanso a través de la pantalla agitando gigantescos machetes al son de un insistente acompañamiento musical. Con la palanca de mando se controla una diminuta varita mágica que vuela dentro de la pantalla destruyendo a los ogros y obteniendo puntos antes de regresar a su mano.

Colocando la varita en el espacio deseado y pulsando el botón de la palanca de mando se pueden plantar árboles en un intento por crear un bosque que le será de ayuda más adelante en el juego. El jugador debe proteger los árboles de los machetes de los ogros y las arañas de Tetragorn, estando atento a sus fuerzas, que se ven mermadas por la picadura de una araña o reforzadas por la muerte de otra. Después de cinco niveles de juego, termina el primer acto con el ataque de las arañas, que agotan rápidamente las fuerzas del druida. El programa entonces congela la acción, cuenta la cantidad de árboles que el jugador ha logrado hacer crecer y sitúa al druida en el siguiente acto.

Segundo acto. Las criptas: Es aquí donde se incuban las arañas. Hay cinco niveles diferentes, cada uno de los cuales contiene dos capas de cuatro criptas. Dentro de las criptas hay huevos de araña, que se iluminan con distintos colores intermitentes cuando éstas están a punto de salir del cascarón. En pantalla también sale el "cofre de los árboles" (que contiene los árboles que se hicieron crecer en el primer acto). El jugador tiene que utilizar su varita para liberar un árbol y desplazarlo a un punto situado sobre una de las criptas; si es rápido, el árbol echará raíces y se abrirá paso a través del techo de la cripta para destruir los huevos antes de que se incuben.

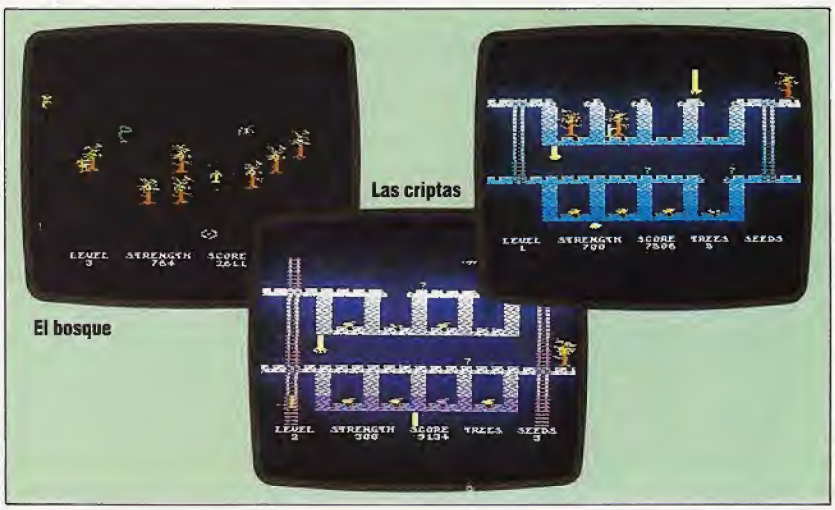
Existe un peligro adicional representado por las "manos del destino"; éstas se estiran desde el techo hacia abajo y cogen todo cuando encuentran debajo de ellas: ya sea el druida, árboles o signos de interrogación. Estos últimos se utilizan para representar premios misteriosos, y debe obtenerse uno de ellos para poder descender por una escalera hasta el siguiente nivel.

Tercer acto. La guarida del nigromante: El clímax de la obra tiene lugar en un cementerio espectralmente oscuro. Las lápidas marcan las muchas tumbas del Necromancer y se deben destruir para impedir su reencarnación. La varita mágica es lo suficientemente poderosa como para matar a cada reencarnación y las propias lápidas desaparecen si uno consigue colocar al druida sobre ellas. Pero la batalla entre el bien y el mal no es así de sencilla, porque todas las arañas que se escaparon en el segundo acto se transforman en arañas "zombis" y acuden en defensa del Necromancer.

Es una cuestión prioritaria reservar muchísima fuerza para la Guarida, porque aquí el jugador debe valerse de todo su ingenio. Ciertamente el último acto puede resultar muy frustrante, dado que no se puede acceder al mismo independientemente del resto del juego.

La de Atari es la mejor de las dos versiones del juego. La versión para el Commodore es más lenta y contiene gráficos y sonido ligeramente menos espectaculares. Ambas pueden verse deslucidas por palancas de mando de respuesta torpe y lenta. No obstante, dejando de lado estas menudencias, quienes posean cualquiera de las versiones encontrarán que éste es un juego magnífico, aunque caro.

Necromancer: Para el Atari 400/800 (32 K) y el Commodore 64
Editado por: Synapse Software, 5327 Jacuzzi St, Suite 1, Richmond, CA 94804
Autores: Bill Williams (Atari), Scott y Steve Coleman (Commodore 64)
Palancas de mando: Atari/Commodore estándares
Formato: Disco y cassette



Liz Heaney

Delicia visual
Necromancer combina gráficos sprite con visualizaciones de fondo en alta resolución para producir unas escenas asombrosas en los tres niveles de destreza. Las relucientes copas de los árboles, el rápido vuelo de la varita mágica del druida, el movimiento de las arañas y los ogros y el fuego del nigromante hacen que *Necromancer* sea no sólo un juego emocionante sino también hermoso desde el punto de vista estético

Una lengua muy culta

Ésta es la primera lección sobre el lenguaje assembly del procesador 6809. Este lenguaje lo incorporan micros como el Dragon y el Tandy Color

Pueden resumirse en tres los componentes de un microprocesador: los registros, que son partes de la memoria interna del procesador; la ALU (*arithmetic and logic unit*: unidad arimético lógica), verdadera "fábrica" de hacer operaciones con los datos que están almacenados en la memoria; y una unidad de control que se encarga de que todo proceda según el orden previsto y en el momento adecuado.

En su más elemental grado de operatividad, el microprocesador sólo responde a señales de voltaje procedentes de sus conexiones con el exterior y causantes de cambios internos (el contenido de los registros) o de ulteriores envíos y recepciones de señales. Si representamos por un 1 la presencia de voltaje y por un 0 la ausencia de éste, entonces podemos dar valores numéricos binarios a estas señales que van y vienen por la memoria y por todo el procesador. Posteriormente se puede programar el procesador numerando las instrucciones según un determinado orden. Así pues, en el más ínfimo nivel de programación, es imprescindible pensar en términos binarios (o hexadecimales). Ello requiere conocer el efecto de cada número, o código de instrucción en el procesador.

Un procesador de ocho bits como el 6809 puede enviar y recibir números binarios de ocho bits, lo que equivale a una gama de números decimales entre el 0 y el 255. Muchos de los números sirven para indicar las direcciones de las posiciones de memoria, las cuales suelen representarse en la mayoría de estos procesadores de ocho bits con números de 16 bits (por lo que podemos numerar posiciones de memoria entre el 0 y el 65535). Es claro que a la hora de tratar estas direcciones de 16 bits, el procesador lo hace de ocho en ocho bits.

Los registros del 6809

Los registros en un procesador pueden adquirir muchas formas según sus funciones. Algunos quedan reservados para el uso interno del procesador, inaccesibles para el programador. Pero cuatro de ellos serán utilizados por éste muy extensamente.

El más empleado es el *acumulador*. La mayoría de los datos se almacenan en él y en él se manipulan. Por ejemplo, la función realizada por una instrucción en assembly como AND es la de sumar el contenido de una determinada posición de memoria al contenido del acumulador. De esta manera "se acumula" un nuevo valor en este registro.

El *registro índice* sirve para modificar direcciones, lo que permite fáciles recorridos por tablas y listas de datos. Cuando una instrucción se refiere a una posición de memoria por el método del *direccionamiento indexado*, el contenido de este registro va a agregarse al de la dirección dada, con lo que se obtiene la *dirección efectiva* de los datos que necesitamos. Para recorrer una tabla de datos, basta con indicar la *dirección de base* (o sea, la del primer valor de la tabla) e irle añadiendo el contenido del registro. Tales registros índice suelen ser de 16 bits, debido a que los valores que en ellos se almacenan son normalmente direcciones.

El *índice de la pila* es el registro que señala la posición de la parte superior de la pila, para almacenar y tomar datos de ella de la manera más rápida. La pila se emplea para ocasiones en que se necesita preservar el contenido interno del procesador (p. ej., al llamar a una subrutina) de modo que no se pierda y pueda recuperarse más tarde. El contenido de algunos, si no todos, los registros puede "meterse" *(push)* en la pila y "sacarse" *(pop)* fuera más tarde, cuando el programa principal recupera el control. El índice de la pila se limita a facilitar al procesador la posición que ocupa el último valor introducido en la pila y dónde se puede colocar o extraer el valor siguiente. También este registro, al tratar con direcciones de memoria, es por lo general de 16 bits.

El cuarto registro es importantísimo, aun cuando su función sea automática casi siempre. Es el *contador del programa*, destinado a guardar la dirección del programa donde se halla la siguiente instrucción a ejecutar. El procesador accede a la posición que le indicó el registro, toma el contenido de esta posición, interpreta su significado y actúa. El contador del programa suele incrementarse automáticamente una vez ejecutada la instrucción, lo que permite tomar las instrucciones una a una y ordenadamente. Si se altera el contenido del contador (porque se almacena un nuevo valor en él, o se le suma o resta un cierto valor) se cambia el flujo del programa. Es decir, funciona como una instrucción GOTO, conocida a este nivel como un "salto" (JMP; inglés: *jump*) en caso de colocarse una dirección completamente nueva, o como una "bifurcación" (BRA; inglés: *branch*) si sólo se alteró con una operación aritmética la dirección que contenía.

Hay todavía un quinto registro, pero no opera del mismo modo que los anteriores. Es el registro de *código de condición*, al que se ha de imaginar más bien como un grupo de bits individuales, cada uno de los cuales representa alguna característica del estado del procesador. Por ejemplo, uno de estos bits indica si el número resultante de una operación realizada dentro de alguno de los registros es cero. Así, es posible recorrer una tabla de valores

Los populares
Las dos máquinas más populares del 6809 son microordenadores: el Dragon de 32 K y 64 K, y el Tandy Color. Existen además numerosos sistemas desarrollados funcionando en universidades y escuelas técnicas europeas. El Tandy Color y los dos modelos Dragon son internamente muy parecidos, y aunque la Dragon Data ha abandonado el mercado, la Tandy aceptó el compromiso de proveer de software y ayuda técnica a los usuarios de Dragon

cargando el número total de valores en un registro y restándole uno al total cada vez que tratemos con un nuevo valor. Cuando el contador alcanza el cero, el bit del código de condición avisa al procesador que no hay más cantidades en la tabla y que puede pasar a otra instrucción. Esto nos permite realizar selecciones (las sentencias IF) y bucles (tipo FOR...NEXT, WHILE...WEND,REPEAT...UNTIL).

Muchos procesadores poseen la *página cero*, integrada normalmente por las primeras 256 posiciones de memoria (en hexadecimal, de la 0000 a la 00FF). Se puede acceder a los valores en ella almacenados con una dirección de ocho bits, lo que hace a la instrucción más corta y más rápida. El procesador 6809 generaliza este concepto incorporando el registro *página directa* de ocho bits, que se encarga de facilitar los ocho bits que faltan para que la dirección sea completa, al hacer referencia a la página cero. Si se cambia el valor contenido en este registro, la página cero puede colocarse en cualquier lugar de la memoria, incluso se puede disponer de más de una página cero.

Un programa en código máquina se compone de una serie de instrucciones entremezcladas con datos y direcciones. Hay personas que pueden programar directamente manejando los respectivos valores numéricos, pero la mayoría de nosotros encuentra esto muy difícil. El lenguaje assembly de un procesador nos permite escribir programas en lenguaje máquina por medio de representaciones mnemotécnicas (para el caso de las instrucciones) y etiquetas (para direcciones y datos). Así, por ejemplo, si queremos cargar en el acumulador los datos de una determinada posición de memoria. Podemos escribir:

```
STORE FCB 0
```

para que quede reservado un sitio en la memoria al que podamos referirnos por STORE, donde colocamos temporalmente un cero. FCB no es realmente una instrucción, sino una directiva que indica al programa traductor del assembly a código máquina que tiene que sustituir una dirección determinada siempre que encuentre la palabra STORE. Más tarde, cuando el programa desee cargar en el acumulador el valor almacenado de ese modo, se podrá utilizar la instrucción:

```
LDA STORE
```

que tomará el valor almacenado en la posición de memoria referenciada por STORE y lo cargará en el acumulador.

El lenguaje assembly de los programas debe ser traducido previamente a su ejecución, para lo cual se utiliza un programa específico llamado *ensamblador*. No tiene éste por qué ser complicado, pues se trata de una relación biunívoca, de uno a uno, entre las sentencias del lenguaje assembly y sus equivalentes en lenguaje máquina. Todo lo que hay que hacer es practicar la sustitución con cuidado para determinar qué valores o direcciones corresponden a qué nombres.

En la próxima lección estudiaremos la estructura interna del procesador 6809 con mayor detenimiento, iniciando ya el análisis de las instrucciones que tendremos a nuestra disposición.

Nuevos horizontes

Juego integrado
Xchange de Psion es un juego de paquetes de software de gestión integrados basado en el software creado para el Sinclair QL. *Xchange* incluye el procesador de textos Quill, el administrador de base de datos Archive, el planificador financiero Abacus y el sistema Easel para gráficos de gestión. Los cuatro paquetes se pueden adquirir juntos o por separado. *Xchange* está a la venta para el IBM PC y PC XT, ACT Apricot y Apricot XI, y Sirius I. Se están planeando versiones adicionales para el Apple Macintosh y el Digital Rainbow

Recientemente Psion ha diversificado su producción, introduciéndose en el campo del software de gestión

Psion fue fundada en 1981 por David Potter, catedrático del Imperial College de Londres. El primer golpe de marketing de la empresa fue un grupo de cuatro paquetes para el recientemente lanzado ZX81: *Flight simulator* (simulador de vuelo), *Backgammon, Vu-Calc* y *Vu-File*, todos los cuales los escribieron Charles Davies y Colly Myers. Esta pequeña gama de programas de calidad (un paquete de simulaciones, un juego, una hoja electrónica y una base de datos) establecieron inmediatamente el nombre de la empresa. En 1982, cuando Sinclair Research encargó un paquete para demostrar el poderío del Spectrum, no fue ninguna sorpresa que la firma de software elegida para desarrollar la cassette fuera Psion.

La popularidad de las máquinas Sinclair ha abierto un enorme mercado para el software de Psion. La empresa ha tenido algunos éxitos notables: la venta combinada de las versiones del Flight Simulator para el ZX81 y el Spectrum, por ejemplo, ha sobrepadado los 500 000 ejemplares. Las estimaciones relativas al total de las ventas mundiales de cassettes Psion han superado la cota de los tres millones. Y con su reciente anuncio de la gama de software "Xchange", Psion ha dado una primera muestra de su deseo de diversificarse, en este caso haciendo un esfuerzo por conseguir parte del mercado de software de gestión.

Psion siempre ha sido una innovadora. Abrió el camino para desarrollar la técnica del software "de compilación cruzada" para ordenadores personales, proceso en virtud del cual se desarrolla un programa en una máquina para utilizarlo en otra. El Horizons, paquete de introducción para el Spectrum, se escribió en un Tandy TRS-80. En la actualidad la empresa utiliza dos miniordenadores VAX 750 para todo su software (véase p. 861). Fue en estas máquinas en las que se escribió el juego de programas del QL: Abacus (una hoja electrónica), Archive (una base de datos), Easel (un programa para gráficos) y Quill (un procesador de textos). Psion Support ha organizado QLUB, un servicio de consulta para usuarios del QL que garantiza una respuesta por escrito a cualquier consulta en 48 horas.

Existen planes ya avanzados para numerosos programas de gestión para los ordenadores IBM PC, Apricot, Sirius, DEC Rainbow y Macintosh. Al igual que el juego de programas del QL, la gama de programas *Xchange* configura una hoja electrónica, una base de datos, un programa para gráficos y un procesador de textos. Lo que diferencia a estos paquetes de otros paquetes de gestión, según Psion, es la sencillez con la que se puede transferir datos entre ellos.

Ian McKinnell

Otro de los desarrollos de Psion que ha sido objeto de muchísima atención es el Organiser (véase p. 921). La aparición de un ordenador de bolsillo proveniente de una empresa que hasta entonces se había considerado como de software constituyó una sorpresa para muchas personas. Un portavoz de la firma, Robin Kinnear, enfoca este desarrollo desde otra perspectiva: "La clave es que Psion *es* una empresa de software para microordenadores. Nosotros pensamos que el Organiser era una idea muy atractiva desde el punto de vista del *empaquetamiento* de software y observamos a nuestro alrededor y descubrimos que no existía nada parecido. De modo que Psion decidió hacerse su propio hardware. El desarrollo estuvo en gran parte determinado por el software".

En la actualidad existen sólo tres paquetes de aplicaciones para el Organiser (ciencias, matemáticas y finanzas), junto con los paquetes de RAM (o *datapacks*, como se llaman) de 8 Kbytes y 16 Kbytes. Psion está considerando la posibilidad de agregar otros paquetes de programas; asimismo, diversas personas que desean escribir software para la máquina se han puesto en contacto con la empresa.

La expansión de mercado es otra de las prioridades de la empresa en estos momentos: recientemente se han establecido subsidiarias en Estados Unidos y Sudáfrica, y se han firmado contratos para la distribución de los productos de Psion en toda Europa. Además, Sinclair Research ha lanzado una gran campaña de ventas en Europa Oriental, empezando con la exportación de 400 ZX81 a Checoslovaquia. Psion, que ya se ha tomado cierto trabajo en la producción de versiones de su software en lenguas foráneas, está segura de seguir avanzando.

Catedrático y empresario
David Potter es el fundador y principal accionista de Psion. Anteriormente académico especializado en Computational Physics en el Imperial College de Londres y en la Universidad de California, fundó Psion como Potter Scientific Investments

Hecho a la medida

La ergonómica es la ciencia que trata de adecuar el ambiente laboral al trabajador. En ella inciden disciplinas tan diversas como la ingeniería y la psicología

La introducción de los ordenadores personales en los entornos doméstico y laboral hace que las consideraciones sobre la ergonómica revistan suma importancia. El color y el sonido constituyen buenos ejemplos respecto a cómo influye el ambiente que rodea a las personas en su rendimiento en el trabajo. Algunas centralitas telefónicas ofrecen música a quienes llaman para relajarlos y tranquilizarlos durante la espera. Una pantalla VDU verde resulta más agradable y relajada a la mayoría de los usuarios de ordenadores, mientras que las pantallas en blanco y negro producen cansancio y tensión.

Diferentes combinaciones de colores provocan respuestas muy distintas: azul sobre amarillo por lo general se considera una visualización atractiva, mientras que cyan (azul claro) sobre verde no suele agradar. El color de una habitación puede influir en el estado de ánimo de la persona que esté en ella: los amarillos se consideran "alegres", los azules y verdes "relajantes" y el esquema de una oficina típica, marrón y gris, "deprimente".

Las sutiles diferencias de este tipo en cuanto a preferencia podrían parecerle irrelevantes o intrascendentes al usuario de un micro personal, pero los especialistas en ergonómica han demostrado repetidamente que cuando las personas se sienten insatisfechas con su entorno (con frecuencia sin saber que es el esquema de color o el ruido de fondo lo que les está afectando), son más propensas a la fatiga visual y el dolor de espalda. Los usuarios de ordenadores sufren comúnmente las consecuencias de la disposición irracional de las habitaciones y las estructuras de trabajo. Un importante estudio ergonómico sobre una oficina en la cual los operadores trabajaban en terminales acomodados junto a las paredes, atribuyó el bajo rendimiento y el elevado porcentaje de errores al aislamiento social y la tensión producidos por la distribución. Se descubrió que cuando los trabajadores se colocaban frente a frente en pantallas de perfil bajo y en medio de la habitación, toda la atmósfera se avivaba y la calidad del trabajo mejoraba.

La introducción de los ordenadores frecuentemente ha tenido un efecto de "pérdida de experiencia" en los trabajos, haciendo poco exigente, rutinario y carente de interés el trabajo de las personas. Ahora ya se reconoce como parte de la misión del analista de sistemas el asegurar que las tareas asignadas a las personas después de la instalación de un ordenador sean satisfactorias y suficientemente exigentes. Para este análisis se necesita el consejo del especialista en ergonómica.

Un diseño de sistema verdaderamente ergonómico considera al operador humano como un importante componente del mismo, con sus propias características operativas y sus tolerancias. Tanto las consideraciones humanitarias como la lógica financiera reconocen que los sentimientos, las percepciones y los hábitos de las personas son tan importantes para la eficacia de un sistema como la capacidad del bus de direcciones o la velocidad del reloj.

Ian McKinnell

El primer contacto
En general, la interface para el usuario empieza en el teclado, que ha sido objeto de muchas mejoras ergonómicas, desde las teclas esculpidas y los teclados cóncavos hasta los teclados numéricos y las visualizaciones LCD. Pero el principal defecto del teclado, el de la ineficacia del trazado QWERTY, parece imposible de erradicar, porque la mayoría de las personas se muestran reacias a aprender nuevas pautas de mecanografía. Las emociones típicamente humanas de esta naturaleza y la aparente irracionalidad con frecuencia constituyen los mayores obstáculos con que se topa la ingeniería de los factores humanos

El programador también se puede beneficiar del estudio atento de los problemas y las exigencias particulares de su trabajo. La programación exige una atención constante, un método lógico y un concienzudo cuidado al detalle, pero son pocas las personas que exhiben naturalmente estas cualidades y la mayoría de ellas se resiste a su imposición. Ésta es una de las razones por las cuales es imposible garantizar programas libres de errores.

La aplicación de la ergonómica al diseño de nuevos lenguajes de programación y a métodos para el diseño de sistemas es una técnica fascinante que ofrece muchos beneficios tanto a los programadores como a quienes los emplean. Los psicólogos se han pasado años estudiando a las personas como dispositivos procesadores de información y tienen ideas claras (aunque aún incompletas) acerca de las estructuras de memoria, velocidades y capacidades del cerebro. Ellos pueden ayudar a diseñar estructuras de datos y de control de programas con las que las personas se sientan cómodas y las puedan utilizar naturalmente o con escaso entrenamiento.

Las consideraciones psicológicas son, asimismo, una parte de los métodos de entrenamiento, selección y organización dentro de la industria. Más im-

portante aún. los psicólogos pueden asegurar que los diseñadores se concentren primero en hacer que el sistema se adapte a las personas, en vez de dar por sentado que será el usuario quien se haya de adaptar al sistema.

En los últimos años los especialistas en ergonómica han estado estudiando la reacción de las personas ante el software complejo, lo que se ha dado en llamar la *interface para el usuario*. Hasta hace poco, los usuarios de ordenadores han sido profesionales experimentados y sumamente motivados, preparados para aceptar la incomodidad y las molestias y deseosos de adquirir un nivel de competencia técnica como precio a pagar por el poder de la máquina. El usuario de hoy en día, sin embargo, es literalmente una persona común, con una tolerancia muy limitada hacia máquinas temperamentales y exigentes; si tuviera que aprenderse un lenguaje de comandos para bases de datos para poder utilizar un cajero automático, ¡probablemente se llevaría sus negocios a las empresas financieras! Y los usuarios que tienen problemas de interface no son casos aislados. Los sistemas de bases de datos que ponen megabytes de información de administración en cada uno de los escritorios de una oficina están poco y mal utilizados en todas partes, porque para extraer información de ellos se requiere cierta fluidez en lenguajes complicados, como el SQL. Tanto los investigadores como los usuarios confían en que la próxima generación de sistemas *front end* de lenguajes naturales haga que sea posible comunicarse con la base de datos o el modelo financiero en un lenguaje humano corriente.

Existen muchas maneras de ayudar al usuario, incluyendo la provisión de facilidades de "ayuda" (*help*) (véase p. 1006). Las investigaciones en el campo de la inteligencia artificial han llevado al desarrollo de programas basados en el conocimiento que pueden explicar cómo llegan a las decisiones, y se cree que estas técnicas podrían ayudar a la creación de "módulos asesores" para guiar a los usuarios de programas. Este tipo de software "inteligente" plantea, no obstante, una cuestión filosófica: ¿qué es una buena explicación, y para quién? El programador es posible que espere una explicación de las estructuras de datos y los procesos involucrados en un sistema, mientras que el usuario de gestión quizá prefiera un análisis de los medios para alcanzar un fin comercial. Las distinciones de esta clase constituyen problemas semánticos y lingüísticos para los especialistas en ergonómica y los científicos de la informática.

Existe un área, igualmente confusa, del estudio de la interface para el usuario a la que los psicólogos llaman *elaboración de modelos*. Las personas utilizan modelos mentales (tales como estereotipos nacionales o racionales) como forma de rellenar vacíos de su conocimiento; la mayoría de nosotros podríamos predecir los puntos de vista, la personalidad y las ideas políticas de un extraño simplemente a partir del conocimiento del trabajo que realiza. Del mismo modo, las personas enfocan los ordenadores con ideas estereotipadas acerca del poder y el comportamiento de la máquina, y pueden considerarlos más inteligentes y mejor informados que seres humanos que realicen la misma clase de tareas. Cuando se encuentran con el tipo de respuestas lacónicas e impersonales que efectúan muchos paquetes de software (especialmente ante entradas incorrectas), tienden a considerar que los ordenadores son antipáticos, incluso hostiles, apreciaciones que están totalmente fuera de lugar. Esta percepción personalizada de la máquina influye sobre toda la interacción, llevando a menudo a respuestas inapropiadas desde ambos lados de la consola.

Los programadores que intenten infundir amabilidad hacia el usuario pueden agudizar el problema al introducir características que hagan que el programa parezca más inteligente de lo que en realidad es. Por ejemplo, utilizar avisos graciosos como HOLA JUAN, SOY EL ESPIRITU DE LA BASE DE DATOS puede estimular al usuario a responder de guisa similar, a menudo con resultados catastróficos y el consiguiente desaliento. La genuina amabilidad hacia el usuario necesita de un enfoque más experto. Implica pensar en las personas y preocuparse por ellas, y dejar un margen para sus complicadas reacciones ante los ordenadores y su trabajo.

Condiciones de trabajo
Está generalmente reconocido que el rendimiento de las personas en el trabajo se ve influido por factores físicos manifiestos, tales como la altura de los escritorios, la temperatura ambiental y los niveles de ruido. Sin embargo, efectos más sutiles de color, luz y percepción del espacio actualmente se están reconociendo como aspectos igualmente importantes del sistema de trabajo, en especial cuando se utilizan ordenadores. Un analista que instale un sistema nuevo debe considerar todos estos factores, además de elegir el hardware y el software

El flexible FORTH

Las personas se pueden sentir frustradas y limitadas por ideas recibidas en la misma medida en que por el diseño de su entorno físico. El FORTH, el lenguaje que muchos consideran un sustituto del BASIC, fue inventado por el astrónomo Charles Moore en el Kitts Peak Observatory de Arizona. Moore estaba trabajando en el control de los movimientos del telescopio utilizando programas en FORTRAN, pero ese lenguaje le resultaba demasiado enfocado a procesos de puro cálculo e inadecuado para aplicaciones de control externo. Encontró la solución a su problema escribiendo un nuevo lenguaje. El FORTH se diferencia de los lenguajes rígidamente estructurados, como el FORTRAN, en que permite que el usuario cree virtualmente una nueva versión del lenguaje para adaptarse a cada nueva tarea de programación. El precio de la flexibilidad del FORTH es su enfoque, severamente hostil.

Mecanismo protector

En este capítulo investigaremos formas de producir una salida desde la puerta para el usuario y esbozaremos una amplia red de control

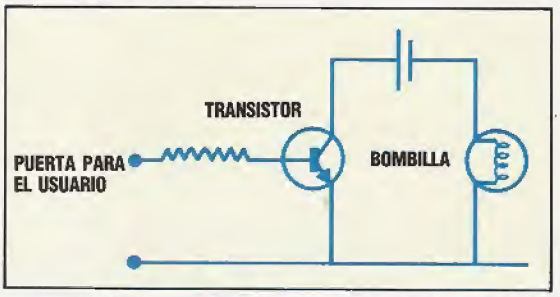
En la mayoría de las aplicaciones de microprocesadores, el término "tampón" *(buffer)* (véase p. 688) ha llegado a significar un lugar de almacenamiento temporal para datos que se están transfiriendo de una parte de un sistema informático a otra. En electrónica analógica, sin embargo, el término se utiliza para describir un circuito que protege a un dispositivo de las acciones de otro. Si deseamos conectar y activar motores eléctricos u otros componentes eléctricos al ordenador para que éste los controle mediante la puerta para el usuario, entonces debemos proteger a los delicados circuitos internos del micro de los componentes a los que lo hemos unido.

El chip de entrada/salida que hay dentro de su microordenador trabaja a niveles de voltaje de 0 y +5 voltios y utiliza corrientes medidas en unos pocos miliamperios (mA). Por lo tanto, hemos de asegurar que no coloquemos voltajes mayores de +5 voltios en ninguna de las líneas de la puerta para el usuario, ni que usemos más de 30 o 40 mA aproximadamente de corriente.

En nuestro capítulo de introducción de este proyecto (véase p. 994) le mostramos cómo poniendo en contacto entre sí los hilos pelados del cable de la puerta para el usuario se podía modificar el contenido del registro de datos. Descubrimos que conectando a tierra las celdas del registro evitábamos la introducción de voltajes o corrientes peligrosas en el sistema y, por consiguiente, no se requería protección para el sistema de circuitos interno del micro. Sin embargo, si deseamos conectar otros dispositivos debemos incluir una protección y ello se puede hacer de diversas formas. Podríamos desear asegurar solamente que no se usaran más de 20 mA de corriente desde cualquier patilla de la puerta. Esto se puede hacer utilizando un relé conectado a la puerta para el usuario para encender y apagar el dispositivo de salida, e insertando una resistencia en el circuito de alimentación del relé. Si el circuito opera a +5 voltios y nosotros deseamos una corriente de no más de 20 mA, podemos emplear la ley de Ohm (voltaje = corriente × resistencia) para calcular el valor de resistencia necesario:

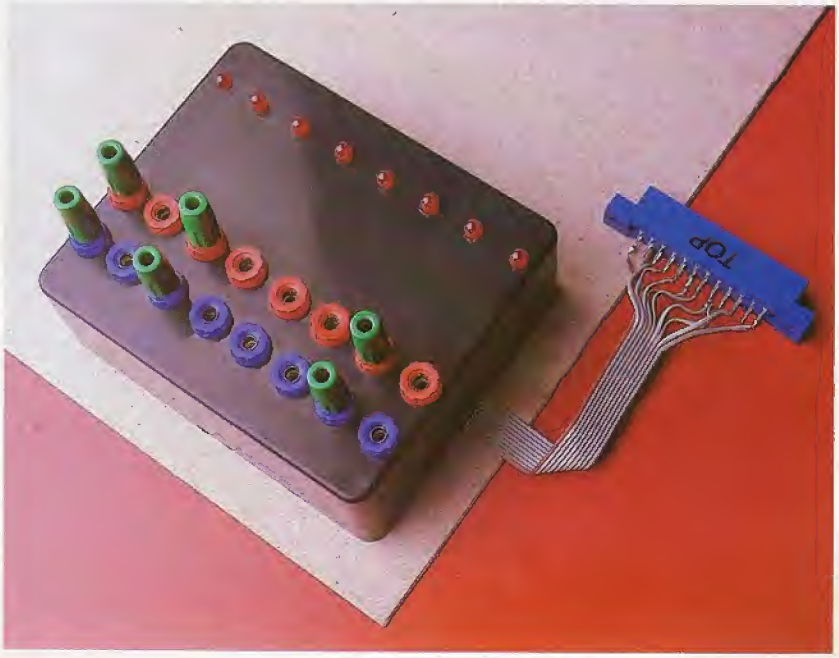
$V = I \times R$
$R = V/I$
$R = 5/0{,}02$
$R = 250$ ohms

Por otra parte, se podría utilizar la salida de una patilla de la puerta para el usuario para disparar un interruptor de transistor de modo que complete un circuito externo:

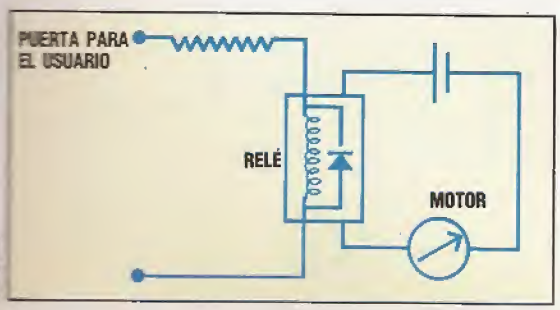


La caja tampón o *buffer* que construiremos utiliza el principio de conmutación del transistor para proteger la puerta para el usuario. Esto es una cuestión de conveniencia, pues los circuitos para las ocho líneas están disponibles todos en un único chip.

Después de conseguido el tamponamiento de la puerta para el usuario podemos seguir adelante para agregarle una serie de módulos que nos permitirán conectar a la misma otros dispositivos. Estos módulos permitirán que controlemos la conmutación de LED, motores de voltajes altos y bajos y relés de la red eléctrica. Entonces seremos capaces



El que va en medio
La caja tampón se conecta a la puerta para el usuario del Commodore 64 o el BBC Micro mediante el cable descrito en el capítulo anterior (véase p. 994). La caja protege al ordenador contra niveles de corriente de entrada/salida peligrosos. Los LED indican el estado de las líneas de salida de la puerta para el usuario, y los enchufes y conectores actúan como interruptores on/off en las líneas de entrada



Liz Dixon

Taladrado de la caja

Los LED y los conectores negros y rojos se montan en la base de la caja, como se ve en la ilustración (pág. 1005). El plástico de la caja es blando y acepta fácilmente la perforación, pero para evitar patinazos y rayaduras cubra su superficie con tiras de cinta adhesiva y después marque los agujeros a taladrar. Marque los agujeros con un punzón o perforación suave y luego taladre: un diámetro de 4 mm para los agujeros de LED, y de 8 mm para los de conector

Nomenclatura componentes

Para construir la unidad de interface serán necesarios los componentes que indicamos aquí. Son todos componentes sencillos que se pueden obtener de cualquier proveedor de artículos electrónicos. Las dimensiones exactas de la carcasa no son decisivas, pero nuestro diseño se ajusta exactamente a la caja descrita.

Cantidad	Artículo
8	Resistencia 4,7 K-ohm 0,4 vatios
8	Resistencia 240 ohm 0,4 vatios
1	Condensador electrolítico 1 µF
1	Condensador 0,1 µF
8	LED rojo
8	Diodo 1N4148
1	Puente rectificador W005
3	Buffer hex 7407
1	Regulador de voltaje µA 7805UC
1	Veroboard 36 franjas 50 agujeros
3	Conector chip DIL de 14 patillas
1	Alambre estañado 2 oz/rollo 22
1	Metro cable plano de 12 vías
8	Conector negro 4 mm
8	Conector rojo 4 mm
8	Enchufe negro 4 mm
8	Enchufe rojo 4 mm
1	Conector de potencia 2,1 mm
1	Conector minicon 12 vías
1	Caja plástica 115×95×37 mm

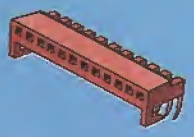
Nuevos componentes

La lista contiene muchos componentes que ya hemos descrito en el curso (véanse pp. 595 y 618), pero los que hemos ilustrado quizá no le resulten familiares. El conector de potencia, el rectificador y el regulador forman parte de la fuente de alimentación eléctrica de la caja. Nuestro diseño nos permitirá utilizar casi cualquier transformador de red eléctrica como entrada, siempre que su salida sea de entre 7 y 25 voltios, CA o CD. El conector de 12 vías se constituirá en la puerta de entrada/salida de la caja a través de la cual los datos pasarán desde y hacia dispositivos externos.

Puente rectificador
Convierte una gama de voltajes de entrada (CA o CD) a voltaje CD de polaridad conocida

Regulador de voltaje
Suaviza la salida del puente rectificador a 5 voltios de continua constante

Conector de potencia
Acepta un enchufe de potencia de 2 mm, como el utilizado en las PSU (*Power Supply Unit*: fuente de alimentación) de muchos ordenadores, y se monta directamente en la placa de circuito impreso

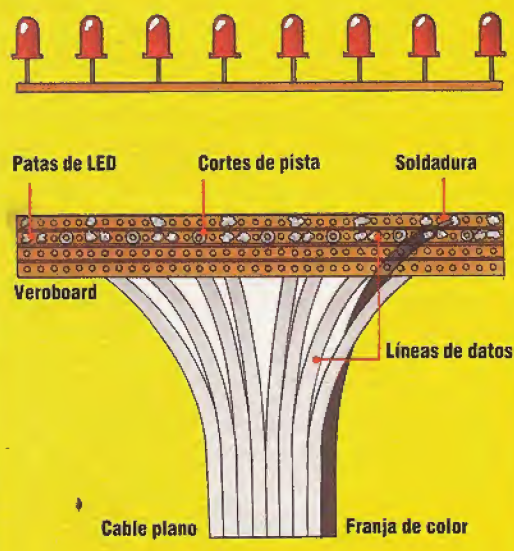


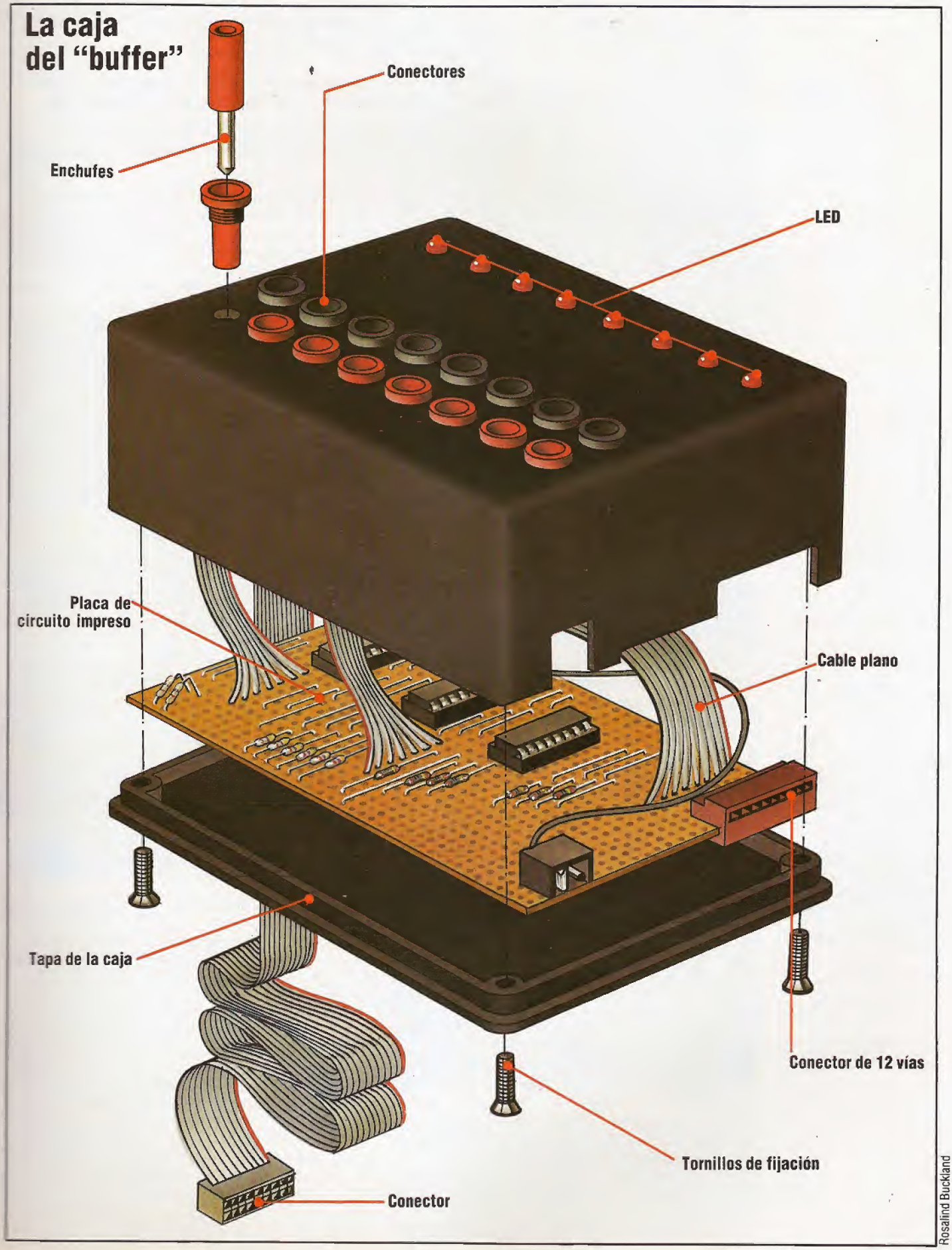
Conector minicon
Se monta directamente en la placa y permite la fácil conexión de los cables externos

de controlar dispositivos domésticos tales como luces, grabadoras de cassette, televisores, etc. Además, podemos agregar un convertidor de digital a analógico, que nos permitirá activar visualizaciones de siete segmentos decodificadas. Debido al poco voltaje y salida de corriente de la puerta para el usuario, necesitaremos también una fuente de alimentación eléctrica externa de nueve voltios. A medida que se vaya construyendo cada módulo, se irá conectando a un bus común, a lo largo del cual se encaminarán las ocho líneas de datos, una línea de tierra y una línea de potencia de nueve voltios. De esta forma podemos poner uno encima de otro o interconectar módulos al sistema. Éste es, pues, nuestro plan de acción para los próximos capítulos de este apartado de *Bricolaje*.

La visualización LED

Los LED ocuparán una franja de veroboard de cuatro pistas de ancho, teniendo cada pista 36 agujeros. Inserte los LED como muestra la ilustración, con las patas más largas en la pista del borde, dejando cuatro agujeros entre cada uno. La espaciación debe ser la misma que la de los agujeros taladrados en la caja; de ser necesario, vuelva a posicionar los LED. Suelde las patas a las pistas de cobre, procurando no arrastrar soldadura de una pista a la otra. Utilice un tester para comprobar la resistencia entre las dos pistas; si es cero, de alguna forma usted ha tendido un puente entre las pistas. Corte un trozo de 20 cm del cable plano de 12 vías y quite tres alambres, dejando un cable de nueve vías incluyendo la de color. Pele y estañe los extremos de los cables. Suelde el cable de color a la pista del borde de la placa. Ahora suelde cada uno de los cables restantes a lo largo de la otra pista, cada cable junto a la pata de un LED. Corte esta pista en siete lugares, de modo que cada par de patilla y cable quede aislado en su pequeña franja de pista. Nuevamente, verifique si hay puentes entre pistas o entre las zonas separadas por cortes. Pase suavemente los LED y la placa por los agujeros de la caja, atornille los conectores en sus agujeros, y después reclínese en la silla y admire su obra mientras espera el siguiente capítulo, en el cual le enseñaremos a construir la placa de circuito impreso





Rosalind Buckland

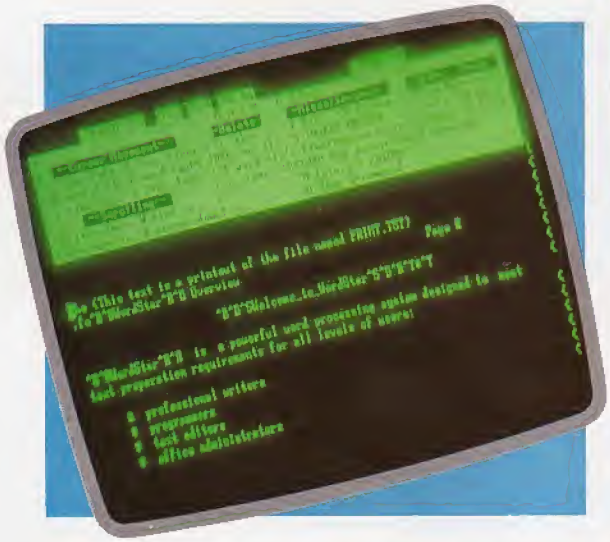
Líneas maestras

En esta ocasión analizaremos el diseño y la implementación de rutinas de "ayuda", que puede incorporar a sus programas

En nuestros días la memoria es barata. La próxima generación de ordenadores personales, que puede que tengan un *mínimo* de 128 Kbytes de RAM, nos permitirá disponer de mayor cantidad de memoria de la que jamás podría exigir ninguno de nuestros más ambiciosos programas. A través de la historia de la informática, la escasez de memoria ha sido la principal excusa que impedía proporcionar a los usuarios suficiente ayuda en forma de instrucciones, mensajes de error sensatos o ayuda en línea. En la actualidad ya no hay ninguna excusa.

Son tres las principales ayudas para el usuario que se pueden proporcionar en un programa: indicaciones de qué hacer, páginas de "ayuda" (*help*) y "postes indicadores". Las indicaciones o instrucciones asumen dos formas. Se pueden ofrecer en un único bloque al comienzo del programa, o se las puede ir suministrando a medida que se las requiera a lo largo del programa (como los avisos para una entrada del usuario, p. ej.). Lo ideal sería que el usuario pudiera tener ambas a su disposición.

En su forma más simple, las instrucciones podrían ser sencillamente una página (o varias) de texto explicando en español llano cómo utilizar el programa. El texto se puede almacenar en series o en sentencias DATA dentro del programa, visualizándose cuando así se solicite mediante una llamada a una subrutina escrita a tal fin. Al comienzo del programa se le pregunta al usuario si necesita instrucciones o ayuda; de ser así, se llama a la subrutina. De allí en adelante el resto de rutinas que acepten datos del usuario deben construirse de forma que con una entrada específica ("?" es común, o puede utilizarse "l") active una llamada a la subrutina de ayuda. Es una buena idea crear una instrucción estándar "visualizar ayuda", y modificar las rutinas de entrada de la biblioteca para aceptarla. No olvide modificar cualquier aviso que emplee en sus rutinas, de modo que "Pulse cualquier tecla para más..." se convierta en "Pulse cualquier tecla para más (o "l" para instrucciones)". Ello le facilitará un formato estándar para todos sus programas.



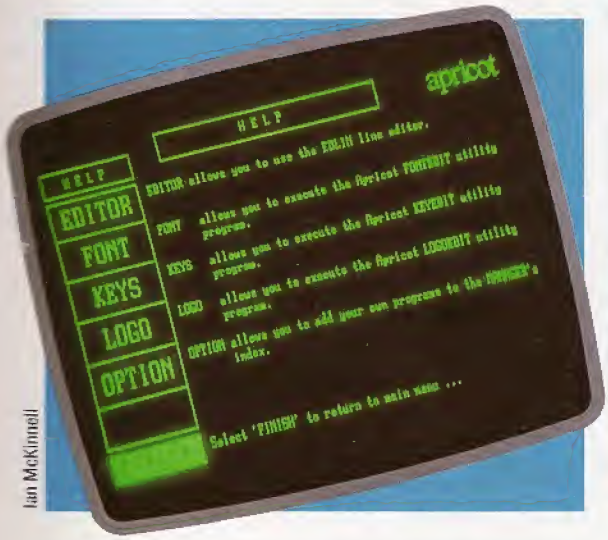
Palabras de consejo
El Micropro Wordstar ofrece un ejemplo muy vendido de software conducido mediante comandos con "ayuda en línea". El usuario puede abreviar el menú de ayuda en pantalla o bien prescindir directamente de él, pero mediante la sola pulsación de una tecla siempre se dispone de una estructura del archivo Help (de ayuda) sumamente detallada

Pero no es necesario que las ayudas se compongan sólo de texto. Se pueden incluir diagramas y desarrollar las rutinas de ayudas o instrucciones como para que ofrezcan ejemplos sencillos y permitan que el usuario practique. Tales rutinas de ayuda son comunes en los programas que efectúan experimentos científicos; en este caso al usuario se le puede exigir que realice una tarea especificada con un determinado nivel de destreza antes de permitirle seguir con el programa principal. Estas rutinas de "enseñanza" no son fáciles de desarrollar porque deben simular el comportamiento del resto del programa, así como evaluar el rendimiento del usuario.

Siguiendo un estilo similar, se podría llamar a las páginas de "ayuda" (*help*) para explicar la operatoria de determinadas partes del programa. Esta facilidad existe en muchos sistemas, que disponen de ella para explicar la utilización de las instrucciones; el sistema operativo Unix, por ejemplo, ¡permite acceder a todo el manual del usuario como ayuda en línea! Facilitar ayudas en sus propios programas no tiene por qué ser más difícil que proporcionar instrucciones: en cada punto apropiado, permita simplemente que el usuario entre una solicitud de ayuda en vez del input normal; cuando ello suceda, el programa habrá de llamar a la correspondiente rutina de ayuda. Es probable que un programa complicado requiera una gran cantidad de páginas de ayuda, de modo que, nuevamente, es deseable una rutina general de ayuda. Ésta podría pedir que el usuario entrase un número para identificar la página concreta de ayuda que se requiere. En los sis-

temas basados en disco, las páginas de ayuda se pueden almacenar como archivos separados. La rutina de ayuda creará entonces el nombre de archivo adecuado a partir de la entrada del usuario, leerá el archivo y lo visualizará en la pantalla.

Tanto las rutinas de ayuda como las de instrucciones bien pueden ocupar más de una simple página de información. De ser éste el caso, se debe diseñar la rutina de visualización de modo tal que el usuario pueda ir hojeando las páginas hacia atrás y hacia adelante a voluntad. También debe asegurarse que el usuario pueda abandonar la rutina y regresar al punto exacto en el cual salió del programa principal: ¡es muy frustrante tener que pasar por 10 páginas de información innecesaria cada vez que se necesitan instrucciones! Si se había dado un aviso, ahora se habrá perdido, y deberá repetirse. La rutina de ayuda debe establecer un indicador que le diga a la rutina llamada que debe volver atrás hasta la última instrucción previa a la llamada de ayuda, limpiando primero el indicador.



Ian McKinnell

Una metáfora común para las interacciones del usuario con programas complejos es la de pensar que éste va navegando a través de una enmarañada red de lógica. El recién llegado al programa no comprenderá su estructura y se puede desorientar y extraviar fácilmente. De modo que se necesitan "postes indicadores" para guiar al usuario. Un menú es el ejemplo más sencillo; opera como una señal vial que muestra las posibles salidas de un cruce. Sistemas como el Macintosh y el Lisa de Apple trabajan de forma similar, utilizando iconos en lugar de opciones de menús.

Algunas direcciones son más importantes que otras. En un sistema basado en instrucciones de ayuda puede haber docenas de posibles instrucciones. Sin embargo, no todas ellas serán relevantes o siquiera posibles en un determinado punto del programa. Si la cantidad de opciones es reducida, es útil visualizar una línea o dos para explicar qué son. Algunas opciones (como QUIT: salir) deben estar disponibles en todo momento, de modo que es una buena idea mantener éstas en pantalla. También pueden estar disponibles constantemente ciertas instrucciones específicas de las aplicaciones. Una técnica común consiste en programar tales instrucciones en teclas de función y visualizar un mensaje de una sola línea para señalar la función de cada tecla. Siempre es una buena práctica visualizar un poste indicador que señale la forma de salir de un programa: esto les da mayor confianza a los novatos, ¡para quienes la principal preocupación suele ser encontrar la salida de emergencia!

Buena gestión

El software Manager del ACT Apricot va guiando al usuario a través de un juego de hostiles programas de utilidades mediante su sistema de menús jerárquicos. La ayuda (Help) es una opción de todos los menús y consiste en una explicación de todas las otras opciones del menú. Éste es un buen ejemplo de software clásico guiado por menú apoyado por grandes archivos de ayuda

Se han desarrollado algunos sistemas experimentales que pueden comprobar el rendimiento de un usuario y ajustar el nivel de ayuda ofrecido en función de dicho rendimiento. Los programas comerciales que posean esta característica todavía están muy lejanos, pero es posible utilizar técnicas sencillas para alcanzar al menos una parte de este objetivo. Si se solicita al usuario que dé su nombre cada vez que se ejecuta el programa, entonces se puede llevar un archivo de usuarios y de sus niveles de destreza. Estos niveles se pueden calcular (a partir de la cantidad de veces que un determinado usuario ha ejecutado el programa o, por ejemplo, a partir de la máxima puntuación conseguida si el programa en cuestión es un juego) y actualizar al final de la ejecución del programa. A medida que aumente el nivel de destreza, se modificará el tipo de ayuda y la señalización con postes, haciéndose más concisos y menos intrusos. Al usuario también se le puede solicitar que elija el nivel de ayuda requerido, como en el paquete para tratamiento de textos Wordstar. En un plano ideal se utilizarían ambas alternativas.

La incorporación de ayuda puede ser una valiosa guía para mejorar el rendimiento de un programa. Una vez que se ha diseñado una rutina de ayuda es muy sencillo modificarla para que registre qué páginas de ayuda se emplearon y con qué frecuencia se las necesitó. Ello proporciona una clara indicación de los focos problemáticos del programa.

Respuestas adecuadas

Soluciones a los ejercicios propuestos en capítulo anterior (p. 996)

Todos los ejercicios de programación sugeridos se parecían al programa ejemplo "Numeros de telefono" en cuanto a controlar la puerta para el usuario y comprobar su estado comparándolo con el de algún valor de referencia. Las soluciones que le presentamos no son, de ninguna manera, las únicas posibles

1) Alarma antirrobo

```
10 REM BBC VERSION 1.1
20 REGDAT=&FE60:RDD=&FE62
30 ?RDD=0:&REGDAT=127
40 REPEAT
50 PRINT "TODO VA BIEN"
60 ALARMA=?REGDAT
70 UNTIL ALARMA <> 255
80 PRINT "ALARMA"
90 FOR PIN=0 TO 7
100 IF (ALARMA AND 2^PIN)=0 THEN PRINT
    "ALARMA, INTERRUPTOR N."PIN
110 NEXT
120 END
```

```
10 REM CBM64 VERSION 1.1
20 RDD=56579:REGDAT=56577
30 POKE RDD,0:POKE REGDAT,255
40 FOR C=0 TO 1 STEP 0
50 PRINT "TODO VA BIEN"
60 ALARMA=PEEK (REGDAT)
70 IF ALARMA <> 255 THEN C=1
80 NEXT C
90 PRINT "SUENA LA ALARMA"
100 FOR PIN=0 TO 7
110 IF (ALARMA AND 2 ↑ PIN)=0 THEN PRINT
    "ALARMA, INTERRUPTOR N.";PIN
120 NEXT:END
READY.
```

2) Contador de impulsos (I)

```
10 REM BBC VERSION 1.2
20 REGDAT=&FE60: RDD=&FE62
30 ?RDD=0:?REGDAT=127
40 CONTADOR=0: REM INICIALIZAR CONTADOR
50 TIME=0: REM INICIALIZAR TIME
60 REPEAT
70 REPEAT
80 UNTIL ?REGDAT <> 255
90 CONTADOR=CONTADOR+1
100 UNTIL TIME > 6000: REM 1 MIN
110 PRINT "IMPULSOS="CONTADOR
120 END
```

```
10 REM CBM64 VERSION 1.2
20 RDD=56579:REGDAT=56577
30 POKE RDD,0:REM TODAS ENTRADA
40 T=TI: REM INIC. TIME
50 IF PEEK (REGDAT)=255 THEN 50
60 CONTADOR=CONTADOR+1
70 IF (TI-T) < 3600 THEN 50
80 PRINT "IMPULSOS=";CONTADOR
90 END
READY.
```

3) Contador de impulsos (II)

```
10 REM BBC VERSION 1.3
20 REGDAT=&FE60:RDD=&FE62
30 DIM C(8)
40 ?RDD=0:TIME=0
50 REPEAT
60 FOR N=0 TO 7
70 C(N)=C(N)+NOT(?REGDAT AND(2^N))
80 NEXT N
90 UNTIL TIME > 6000
100 FOR N=0 TO 7
110 PRINT "LINEA"N"="C(N)
120 NEXT N
130 END
```

```
10 REM CBM64 VERSION 1.3
20 RDD=56579:REGDAT=56577
30 POKE RDD,0:T=TI
40 FOR K=0 TO 7
50 C(K)=C(K)+NOT(PEEK(REGDAT)AND(2 ↑ K))
60 NEXT K
70 IF (TI-T) < 3600 THEN 40
80 FOR K=0 TO 7
90 PRINT "LINEA";K;"="C(K)
100 NEXT K
110 END
```

4) Cerradura con combinación

```
10 REM BBC VERSION 1.4
20 REGDAT=&FE60:RDD=&FE62:FLAG=1
30 ?RDD=15: REM LINEAS 0-3 ENTRADA
40 DIM C(3)
50 REM CODIGO=359
60 C(1)=3:C(2)=5:C(3)=9
70 FOR N=1 TO 3
80 A$=GET$:REM ESPERAR PULSACION TECLA
90 NEXT N
100 IF FLAG=1 THEN PRINT "ABIERTA" ELSE
    PRINT "COMBINACION EQUIVOCADA"
110 END
```

```
10 REM CBM64 VERSION 1.4
20 RDD=56579:REGDAT=56577
30 POKE RDD,15:REM LINEAS 0-3 ENTRADA
40 C(1)=1:C(2)=2:C(3)=4: REM CODIGO 124
50 FOR K=1 TO 3
55 PRINT"ENTRE UN DIGITO"
60 GET A$:IF A$="" THEN 60:REM ESPERAR PULSACION TECLA
65 PRINT A$
70 IF PEEK(REGDAT) <> C(K) THEN FL=1
80 NEXT K
90 IF FL=1 THEN PRINT "COMBINACION EQUIVOCADA":END
100 PRINT "ABIERTA"
110 END
```

5) Cambiar el color de la pantalla

```
10 REM BBC VERSION 1.5
20 MODE5
30 REGDAT=&FE60:RDD=&FE62
40 ?RDD=128: REM TODAS ENTRADA EXCEPTO D7
45 ?REGDAT=255
50 A$=GET$: REM ESPERAR PULSACION TECLA
60 COLP=?REGDAT:REM FONDOS > 127
70 GCOL0,COLP: REM CAMBIAR COLOR PANTALLA
80 CLG: REM LIMPIAR PANTALLA GRAFICOS
90 END
```

```
10 REM CBM64 VERSION 1.5
20 RDD=56579: REGDAT=56577
30 POKE RDD,0: REM TODAS ENTRADA
40 GET A$: IF A$="" THEN 40
50 COLP=PEEK(REGDAT)
60 POKE 53281,COLP
70 END
```

Una buena jugada

El Sega SC3000H es un micro de precio muy asequible, creado por una firma famosa por sus máquinas de juegos recreativos

El nuevo ordenador personal de Sega, el SC3000H, se ha presentado en diversas ferias de informática celebradas recientemente y ha sido objeto de comentarios favorables. Aunque es poco revolucionario en cuanto a diseño o función, utilizando el ya familiar procesador Z80A, se trata de un ordenador personal bien diseñado y fácilmente ampliable con gran cantidad de software disponible.

El Sega, una máquina atractiva y ligera que pesa 1,1 kg, tiene una carcasa plástica negra con teclas alfanuméricas blancas y teclas de operaciones grises (para funciones especiales, Control, Shift, Return, etc.). Posee teclas plásticas moldeadas tipo máquina de escribir, que al pulsarse recorren aproximadamente un centímetro. Las teclas hacen "clic" al ser pulsadas, lo que posiblemente sea un remanente del teclado de goma blanda utilizado en la versión japonesa. En líneas generales, no obstante, la calidad del teclado es buena para una máquina de su precio. Además de las teclas estándares, el Sega posee una tecla de función no programable que se utiliza para entrar palabras clave de BASIC, una tecla *graph* (de gráficos) para acceder a los símbolos para gráficos del teclado, teclas *clear screen* (limpiar pantalla) e *insert/delete* (insertar/eliminar) y un grupo de cuatro teclas para el cursor que resulta ideal para juegos. Sin embargo, un serio inconveniente es la ausencia de símbolos gráficos en las teclas. El SC3000 *soft-key* (de teclas blandas), que sólo se distribuye en Japón, tiene dichos símbolos impresos en las teclas; sin ellos la operación del SC3000H en BASIC resulta más dificultosa.

El SC3000H está bien equipado con interfaces. Contiene dos puertas para palanca de mando estilo Atari situadas sobre el lado izquierdo y una ranura para cartucho de ROM a la derecha, y las conexiones de la parte posterior de la máquina incluyen la salida para un aparato de televisión, una puerta para pantalla de color compuesto, otra para impresora tipo DIN, interface para cassette y conector de potencia para el adaptador de corriente de nueve voltios. También incluye una caja de conexiones para usar con un televisor y un cartucho de BASIC con un pequeño folleto de instrucciones.

La instalación del Sega es directa, lo que está muy a tono con la escasa documentación de ayuda. Un folleto de dos páginas describe la conexión del ordenador a un televisor y a la fuente de alimentación eléctrica. Un LED verde indica que la máquina está encendida, pero en el folleto no se mencio-

Recién llegado de Japón
Por su asequible precio se pretende que el Sega SC3000H entre en competencia con el Sinclair Spectrum y el Commodore 64. La máquina posee una gama completa de periféricos, incluyendo grabadora de cassette, palancas de mando e impresora-plotter en color

Chris Stevens

na que el Sega no funcionará a menos que se inserte en la puerta de ROM un cartucho que contenga BASIC o un juego. Además, hay intérprete interno de BASIC, y después de cargar el cartucho de BASIC el SC3000H queda con unos ridículos 515 bytes de RAM para el usuario: menos memoria que la que proporciona el ZX81 sin ampliar. Ésta es una clara desventaja en el mercado, porque significa que el usuario habrá de adquirir un módulo de ampliación si quiere utilizar la máquina para algo más que ejecutar cartuchos de juegos.

El manual podría asimismo inducir a confusión, dado que se refiere al teclado blando del SC3000 y es necesario que el usuario consulte el diagrama del teclado impreso para situar cualquier símbolo para gráficos. Este inconveniente se podría corregir en versiones ulteriores. El diagrama del teclado también muestra las palabras clave del BASIC impresas arriba o abajo de muchas de las teclas, al estilo del Spectrum. A ellas se accede manteniendo pulsada la tecla de función al mismo tiempo que la tecla alfanumérica que corresponda. De esta forma se puede acceder a instrucciones como RUN, LOAD y GOTO, a funciones matemáticas (ABS, SIN, COS, TAN) y funciones para series como LEFT$, RIGHT$ y MID$; pero, de nuevo, es necesario remitirse al manual constantemente.

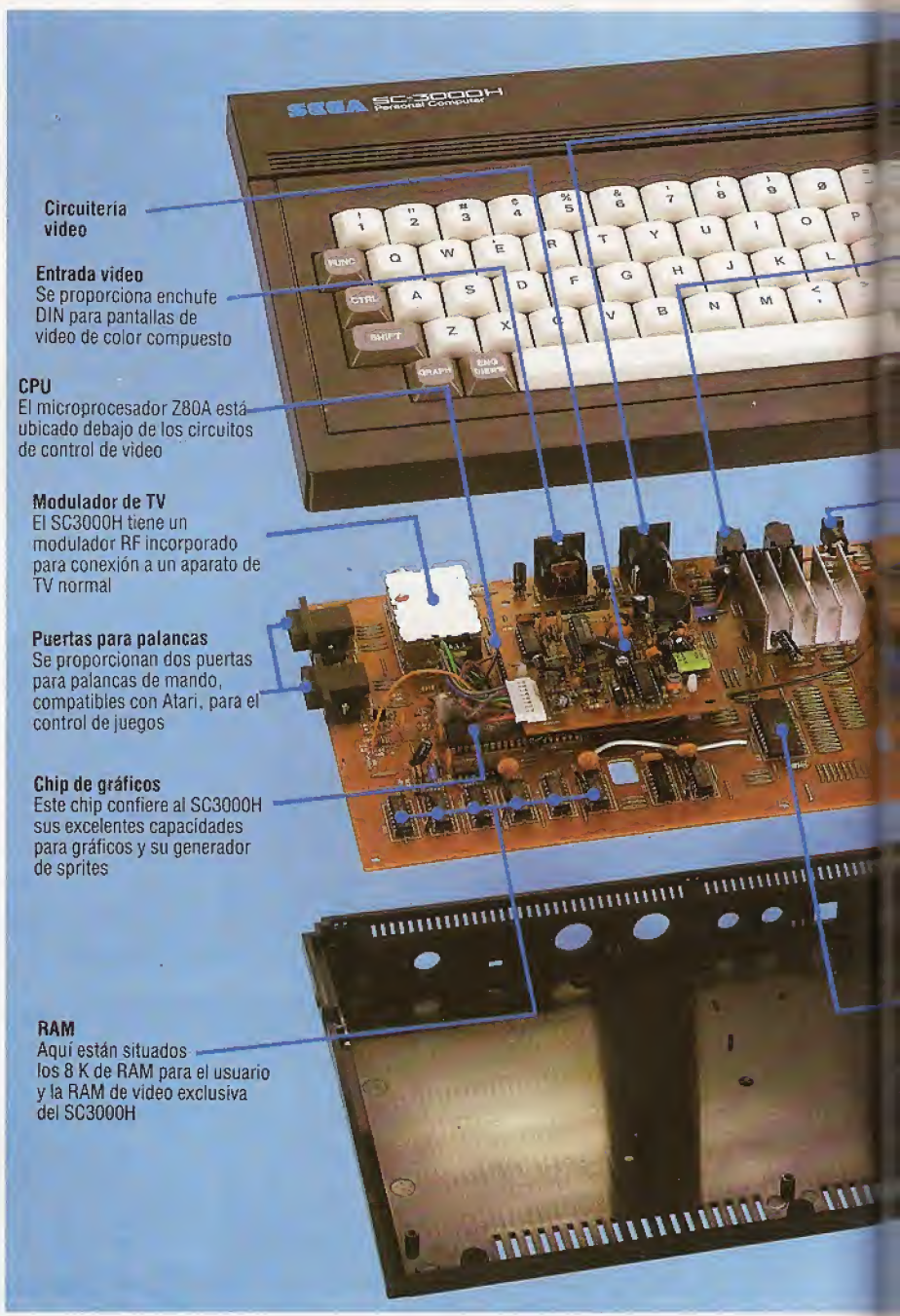
En operación, los gráficos del SC3000H son impresionantes. La visualización configura dos pantallas: la pantalla de texto ofrece 24 filas de 40 columnas en dos colores, mientras que la pantalla para gráficos tiene una resolución de 256 × 192 pixels y es capaz de visualizar 16 colores. Los niveles de brillo se pueden ajustar para proporcionar hasta 210 matices y se pueden crear hasta 32 sprites. El BASIC Sega, que es similar al Microsoft Extended BASIC, utiliza las instrucciones DRAW, COLOR, PAINT y SPRITE para la programación de gráficos.

El SC3000H posee asimismo seis canales de sonido, que se pueden direccionar a través de instrucciones POKE o mediante la utilización de las instrucciones del BASIC BEEP y SOUND. Un cartucho de música sirve de ayuda para la composición, si bien las melodías producidas no exhiben en demasía la "calidad de sintetizador" a que hace mención el manual.

La experiencia de Sega en juegos recreativos se refleja en la calidad de su software de juegos. Los gráficos de éstos generalmente son buenos, aunque el SC3000H parece tener dificultad para visualizar textos y gráficos al mismo tiempo. Visualmente, los juegos guardan mucha semejanza con sus equivalentes recreativos, y la gran calidad del sonido hace que resulten aún más disfrutables. La tecla Reset se utiliza de una manera inusual: en vez de reiniciar un juego, funciona como un interruptor de palanca para introducir una pausa en la acción.

Sega suministra dos tipos diferentes de palanca de mando, pero ninguna de ellas proporciona un movimiento rápido y uniforme. No obstante, el grupo de teclas del cursor, tan sensatamente diseñado, facilita la sencilla operación del software para juegos desde el teclado.

La empresa comercializa una buena gama de equipos periféricos para la máquina, incluyendo una grabadora de cassette, impresora-plotter en color y una unidad de ampliación. Ésta proporciona 64 Kbytes extras de RAM y tiene una unidad de disco compacto incorporada.



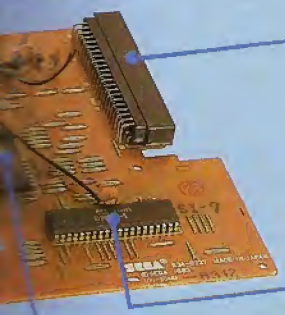
Palancas de mando del Sega SC3000H
Hay dos palancas de mando distintas para el Sega, cada una de ellas apropiada para diferentes estilos de juego. Ambas utilizan un mecanismo de interruptor de ocho posiciones para el movimiento del bastón y son compatibles con el conector estándar Atari de nueve patillas

Chris Stevens

Conector de impresora
Éste es un enchufe tipo DIN para conectar impresora-plotter en color de Sega y otras impresoras

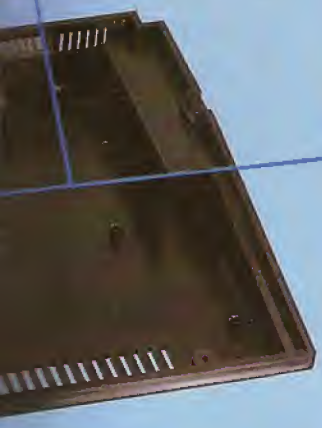
E/S cassette
Si bien la mayor parte del software Sega está basado en cartucho, se proporciona control de cassette para que el usuario almacene programas

Adaptador de corriente



Puerta para cartuchos
La mayor parte del software para juegos de Sega se suministra en cartuchos de ROM que se conectan con el SC3000H a través de esta puerta

Control de E/S
Este chip controla muchas funciones de entrada y salida del SC3000H



ROM
Puesto que el Sega no posee un lenguaje BASIC incorporado, la ROM se utiliza fundamentalmente para operaciones internas y visualizaciones de pantalla

Sinbad Mystery (El misterio de Simbad)
Un juego que combina laberinto y aventuras exóticas

Safari Hunting (Cacería)
Cace feroces animales de la jungla con un fusil tranquilizador

Congo Bongo
Trepe por peligrosas pendientes persiguiendo al gorila. En este loco juego recreativo cuídese de los cocos que caen y de los molestos monos

Baseball
Batee contra el nutrido equipo contrario en esta realista versión

Software Sega
Los gráficos y el sonido del software suministrado para el SC3000H poseen la calidad que cabría esperar de una empresa de juegos electrónicos como Sega. Muchos de los juegos tienen excelentes visualizaciones tridimensionales y constituyen una innovadora reelaboración de los formatos de juegos estándar

SEGA SC3000H

DIMENSIONES
353×210×46 mm

CPU
Z80A, 4MHz

MEMORIA
8 K de RAM, ampliables internamente a 48 K;
18 K de ROM, ampliables internamente a 32 K;
16 K de RAM de video

PANTALLA
24 filas de 40 columnas de texto; resolución de gráficos de 256×192 pixels con sprites y 16 colores en 15 niveles de brillo

INTERFACES
Puerta para cartuchos de ROM, puertas para palancas de mando (2), TV y video compuesto, audio, cassette e impresora

LENGUAJES DISPONIBLES
BASIC, LOGO

TECLADO
64 teclas, tipo máquina de escribir, con juego de cursor, tecla de función, teclas para gráficos y caracteres especiales

DOCUMENTACION
Folleto de instalación de dos páginas y el manual que viene con el cartucho de BASIC. Este pequeño folleto es bastante descriptivo, pero no menciona que la máquina no hará nada hasta que se cargue un cartucho. Toda la documentación se refiere a la versión *softkey* del teclado

VENTAJAS
Se trata de una máquina agradable con notables gráficos y sonido y un BASIC bien equipado. Por su precio, es una buena máquina para juegos y un micro adecuado para principiantes

DESVENTAJAS
La documentación es insuficiente. Los 8 K de memoria hacen que programar sea virtualmente imposible. El teclado debería tener símbolos especiales impresos

El Sega SC3000H es una máquina de agradable uso y contiene algunas características útiles que no son habituales en un micro de su precio. Sega debería tomar medidas para mejorar la documentación, y el teclado necesita un cambio de estilo para que incluya etiquetas para palabras clave y gráficos. El principal inconveniente es, sin duda alguna, la limitada memoria para el usuario; en la actualidad la máquina sólo es apta para emplearla con el software para juegos de Sega: quien desee escribir programas necesitará ampliar la memoria interna o adquirir la unidad de ampliación.

Tortuga inquieta

Veamos cómo utilizar los gráficos tortuga del lenguaje LOGO para dibujar formas complejas con el mínimo esfuerzo

Abreviaturas

Muchas instrucciones de LOGO tienen abreviaturas; he aquí las correspondientes a las instrucciones que hemos mencionado:

FORWARD	FD
BACK	BK
RIGHT	RT
LEFT	LT
PENUP	PU
PENDOWN	PD
PRINT	PR
FULLSCREEN	FS
TEXTSCREEN	TS
SPLITSCREEN	SS

La primera versión del LOGO que salió para microordenadores fue el LOGO MIT; ahora se lo considera la versión estándar y lo produce Terrapin Inc. para las máquinas Apple y Commodore. Logo Computer Systems Inc. (LCSI) produjo otra versión para los ordenadores Apple, Atari y Spectrum, y pronto estará a la venta el LOGO LCSI para el BBC Micro. Existen otras versiones, pero estas dos son las más difundidas. Todos los programas que daremos a modo de ejemplo usan el LOGO MIT; cuando existan diferencias entre las versiones MIT y LCSI, las mismas se explicarán en el recuadro de "Complementos al LOGO".

Sólo hay una forma de aprender LOGO: ¡experimentando! Le sugeriremos algunas cosas para que vaya probando, pero lo mejor que se puede hacer es resolver problemas que uno mismo se haya planteado.

Después de cargado, el LOGO queda en modalidad "inmediata" y listo para recibir y obedecer órdenes (o comandos). En la mayoría de las versiones, estas instrucciones deben entrarse en letras mayúsculas. Digite DRAW y verá que la pantalla se divide en dos secciones (esto se denomina *modalidad de pantalla dividida*). La sección superior es para los gráficos; ocupa la mayor parte de la superficie de visualización, y en el centro se halla la "tortuga", representada mediante un pequeño triángulo. La sección inferior es para texto y de momento simplemente contendrá la indicación "?".

La tortuga es un objeto con el cual nos podemos comunicar dándole instrucciones. Si pensamos que es un "objeto" podremos comprender más fácilmente la programación con ella. Las cosas más importantes a considerar son la posición de la tortuga, hacia dónde apunta o encabezamiento (dirección) y si el "lápiz" que transporta está bajado (en cuyo caso dibujará una línea a medida que se mueva) o levantado (en cuyo caso se deslizará sin dejar ninguna huella). La digitación de DRAW posiciona la tortuga en el centro de la pantalla, mirando directamente hacia arriba y con el lápiz hacia abajo.

Ahora intentemos darle una instrucción:

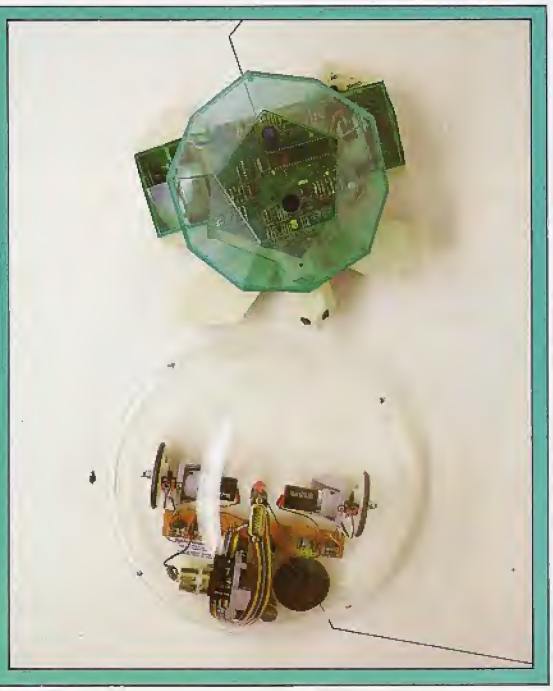
```
FORWARD 40
```

La tortuga se moverá 40 unidades hacia arriba de la pantalla, dibujando una línea a medida que avance. FORWARD es una instrucción de tortuga y el número 40 es su "entrada" (input). Unas órdenes necesitan entradas y otras no; DRAW, por ejemplo, no exige entrada.

Una segunda instrucción de tortuga es BACK (atrás). BACK 10 le indica a la tortuga que se desplace 10 unidades hacia atrás. De modo que FORWARD y BACK (cada una con un número de unidades como entrada) modificarán la posición de la tortuga en la pantalla. RIGHT (derecha) y LEFT (izquierda), por otra parte, no modifican la posición de la tortuga, sino simplemente la hacen rotar; es decir, cambian la dirección en que está encarada. Estas dos instrucciones exigen como entrada un ángulo de entre 0 y 360°. Experimente un poco utilizando estas instrucciones; intente dibujar algunas formas simples; vea lo que sucede si le indica a la tortuga que se

Conozcamos la tortuga

Una tortuga es una herramienta robot de dibujo. Posee ruedas, está controlada por motores paso a paso y tiene un lápiz retráctil. Se la puede instruir para que se desplace hacia adelante, hacia atrás, a izquierda y derecha, y se puede elevar o hacer descender su lápiz. Cuando se baja, el lápiz produce un trazado. Cuando se desarrollaron por primera vez las tortugas tenían forma abombada, como la Edinburgh que vemos aquí, y se controlaban desde un teclado de ordenador. Esta tortuga se conecta al ordenador a través de un cable en paralelo. Los modelos más modernos son a control remoto. Ahora hay a la venta una versión de la tortuga Edinburgh controlada por radio. Y existe un robot con forma de tortuga, la Valiant Turtle, que posee una conexión de infrarrojos con el ordenador. Por extensión, la denominación de "tortuga" también se aplica al cursor de dibujo de la pantalla del ordenador en el LOGO. La mayoría de las tortugas de pantalla son simples formas triangulares, aunque el LOGO Atari visualiza un diminuto cursor en forma de tortuga



Paul Chave

desplace una distancia mayor de lo que permiten las dimensiones de la pantalla; intente utilizar como entradas números negativos. Para volver a empezar con la pantalla limpia, entre DRAW.

Cuando pruebe las instrucciones verá que la tortuga penetra en la sección de textos de la pantalla y, por tanto, parecerá hallarse por "detrás" del texto, de modo que no se la puede ver. Estas instrucciones le serán de ayuda:

FULLSCREEN (pantalla completa); permite usar la superficie total de pantalla para gráficos;
TEXTSCREEN (pantalla de texto): elimina todos los gráficos y deja sólo la modalidad de texto;
SPLITSCREEN (pantalla dividida): vuelve a la modalidad de pantalla dividida;
PENUP (lápiz levantado): permite que la tortuga se desplace sin dibujar ninguna línea;
PENDOWN (lápiz abajo): la tortuga va dejando una "huella" a medida que se desplaza.

Todas las instrucciones que hemos mencionado hasta ahora hacen que la tortuga obedezca sus órdenes. Pero también se puede utilizar el LOGO para obtener información de ella. La dirección a que apunta se mide en grados; un encabezamiento de 0 indica que la tortuga está mirando directamente hacia arriba, y el encabezamiento se mide en el sentido de las agujas del reloj a través de 360°. Para hallar el encabezamiento de la tortuga, digite:

PRINT HEADING (Imprimir encabezamiento)

La posición de la tortuga en la pantalla se define en términos de un sistema de coordenadas, con origen en el centro de la pantalla (es decir, comienza en un punto con las coordenadas *x* e *y* de 0). La posición se puede hallar en cualquier momento digitando:

PRINT XCOR PRINT YCOR (Imprimir coordenada *x* imprimir coordenada *y*)

Pruebe estas instrucciones dibujando una forma y cerciorándose luego de la posición de la tortuga y la dirección hacia la cual está orientada. Utilice estos datos para devolverla a su punto de comienzo.

Quizá se haya encontrado con que mientras experimentaba con instrucciones de LOGO, en la zona de textos aparecían mensajes de error. De no ser así, entonces cometa un error deliberado para ver lo que sucede. Por ejemplo, digite:

FORWARD50

Verá salir el mensaje:

THERE IS NO PROCEDURE NAMED FORWARD50
(No existe ningún procedimiento denominado FORWARD50)

La causa de esto es que el LOGO exige un espacio entre la instrucción FORWARD y la entrada 50, para eliminar cualquier confusión con una posible instrucción llamada FORWARD50. También puede obtener un mensaje de error si digita una instrucción en letras minúsculas.

El LOGO está equipado con un editor de líneas que permite que se corrijan instrucciones si se observa algún error antes de pulsar Return. Utilice las teclas del cursor para desplazarse a lo largo de la línea hasta el texto equivocado. Para insertar caracteres, simplemente digítelos: el texto situado a la derecha del error se desplazará automáticamente para hacer sitio a los caracteres extras. La tecla Delete elimina el carácter situado a la izquierda del cursor. Una vez corregida la línea, pulsando Return el LOGO aceptará la nueva instrucción. Si ya ha pulsado Return antes de detectar el error, digitando Control-P se recuperará la última línea para su edición. Esta característica es igualmente importante para la repetición de instrucciones.

Ahora podemos probar con algo un poco más matemático: por ejemplo, un cuadrado. Recuerde que un cuadrado tiene cuatro lados iguales y que sus esquinas son ángulos rectos (90°), de manera que algo como esto producirá el resultado deseado:

```
FD 50 RT 90
FD 50 RT 90
FD 50 RT 90
FD 50 RT 90
```

Observe que hemos abreviado las instrucciones y que podemos poner más de una en cada línea.

Lamentablemente, llegados a este punto puede que usted se encuentre con un problema técnico: que su cuadrado se parezca más bien a un rectángulo. Ello se debe a la "proporción de tamaños" *(aspect ratio)* de su pantalla; es decir, a la proporción del tamaño de paso vertical y el tamaño de paso horizontal. El LOGO posee una instrucción para manejar esto: utilice ASPECT seguida de un número (el valor por omisión es 0,8) para cambiar la proporción de tamaños hasta que su forma sea realmente un cuadrado. Ahora pruebe con los siguientes ejercicios: dibuje un triángulo equilátero, un pentágono, un hexágono, varios rectángulos, un rombo, un paralelogramo... en fin, ¡todo lo que se le ocurra!

Se podrían simplificar algunas de las instrucciones y reducir la cantidad de digitación mediante el empleo de REPEAT. Para volver a dibujar el cuadrado, digite simplemente:

REPEAT 4 [FD 50 RT 90]

REPEAT (repetir) es una instrucción con dos entradas. La primera es un número que indica la cantidad de veces que el LOGO debe hacer algo, y la segunda es una "lista" de las instrucciones a obedecer. Esta lista siempre debe estar encerrada entre corchetes. De modo que nuestro ejemplo del cuadrado le dice al LOGO que debe repetir cuatro veces la secuencia FORWARD 50 RIGHT 90. Ahora intente simplificar la construcción de las formas que ya haya creado mediante la utilización de REPEAT, y vea si puede producir las formas de estrella que aparecen en algunas de nuestras ilustraciones.

Complementos al LOGO

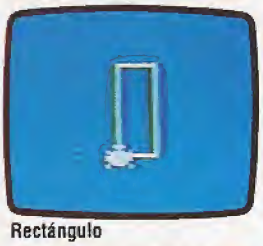
En el Atari, la tortuga tiene aspecto de tal.
En todas las versiones LCSI:
Utilice CLEARSCREEN (limpiar pantalla; abreviatura CS) para empezar a dibujar.
Utilice Control-Y para recuperar la última línea (excepto en el Spectrum, que no cuenta con esta posibilidad).
Para modificar la proporción de tamaños:
Apple LCSI — SETSCRUNCH seguido de la nueva proporción.
Atari —. SETSCR seguido de la nueva proporción.
Spectrum — SETSCRUNCH seguido de dos núms. de coords. [*x*,*y*] (la norma es [100 100])

Ejercicios de LOGO
¿Puede escribir procedimientos para crear estas formas? Las muestras que ofrecemos se dibujaron con LOGO LCSI en un Atari 600 XL


Triángulo equilátero

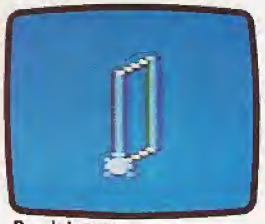

Pentágono


Hexágono


Rectángulo


Estrella de cinco puntas


Estrella de diez puntas


Paralelogramo

Ian McKinnell

Sonidos en secuencia

En este capítulo analizaremos un desarrollo de la secuenciación: la interface MIDI

El secuenciador ha tenido un impacto dramático en la composición musical, tanto en las actuaciones en vivo como en los estudios de grabación. Pero el inconveniente del secuenciador es que puede controlar una gama de variables en un solo sintetizador. Si un músico de teclado posee dos sintetizadores, uno con poderosas capacidades de secuenciación y el otro con buen sonido, no existe forma alguna de conectarlos en una única unidad coordinada. El problema es más grave en los estudios que necesitan coordinar diversos sonidos provenientes de distintas piezas del equipo. La finalidad de una interface digital para instrumentos musicales es la de realizar esta conexión y colocar el control del sistema en manos de una sola máquina de enorme capacidad. La MIDI constituye un intento de conseguir esto en un formato estandarizado, de manera que cualquier sistema digital pueda controlar a otro en la producción de música.

La MIDI (*Musical Instrument Digital Interface*: interface digital para instrumentos musicales) se desarrolló después de una serie de reuniones que celebraron los principales fabricantes japoneses y norteamericanos de instrumentos digitales. Las especificaciones actuales fueron terminadas en agosto de 1983. Está diseñada como un circuito de control para transmitir datos de un instrumento a otro, o desde un microordenador a un instrumento.

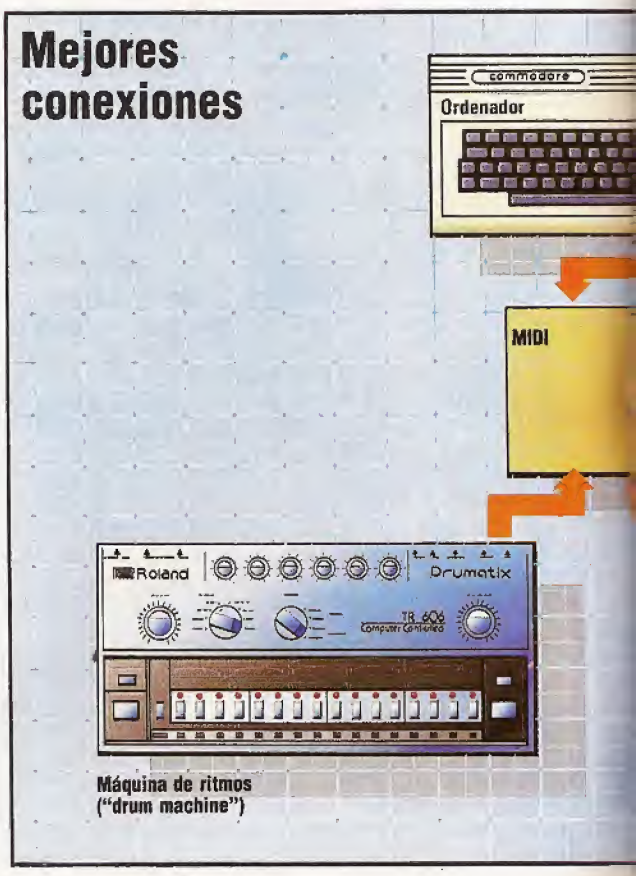
La MIDI opera con un procesador de ocho bits. Sus diseñadores tuvieron que elegir entre los dos medios de transmisión disponibles: en paralelo o en serie. La transmisión en parelelo proporciona líneas individuales, de modo que los ocho bits de cada byte de datos se envían simultáneamente y, por consiguiente, permite una gran velocidad de transmisión. Su desventaja reside en el costo extra que implica y el inconveniente de requerir al menos ocho líneas cableadas en un conector tipo D de 25 patillas. La transmisión en serie sólo utiliza dos líneas. En una los bits de datos se envían uno después del otro; se proporciona una segunda línea para que el instrumento receptor avise los errores de paridad (véase p. 667) en la corriente de datos que vuelve al instrumento maestro o microordenador. Así, la transmisión en serie es más lenta que la transmisión en paralelo, pero tiene la ventaja de un conector más sencillo y de ser mucho más barata.

La razón de ser original de la MIDI fue proporcionar un circuito de control que pudiera realzar la música sintetizada y proporcionar alguna forma de control de altura (*pitch*). Su índice de comercialización había de estar situado en una gama de precios asequible a la mayoría de los usuarios de ordenadores personales y de los músicos que utilizaran sintetizadores y máquinas de ritmos. Fundamentalmente fue éste el motivo por el cual se adoptó para la MIDI la transmisión en serie.

La MIDI transmite de forma asíncrona, a lo largo de las mismas líneas que la interface RS232, que es estándar para conectar muchos microordenadores con modems o impresoras en serie. Cuando se transmiten datos de forma asíncrona, ha de definirse cada byte para el instrumento receptor. En la MIDI esto lo efectúa un chip ACIA (*Asynchronous Communications Interface Adaptor*: adaptador para interface de comunicaciones asíncronas) Motorola 6850, que agrega dos bits extras a cada byte del instrumento maestro. Empieza en "0", el bit de comienzo (*start*), seguido de los ocho bits de datos en serie, y termina en "1", el bit de final (*stop*). Esta palabra en serie de 10 bits se transmite entonces al instrumento receptor, donde un segundo chip ACIA lo vuelve a convertir en los ocho bits de datos reales.

El costoso chip ACIA está protegido dentro del circuito mediante optoaislamiento. Un *optoaislador* en un dispositivo que utiliza células fotoeléctricas para permitir que dos circuitos no conectados intercambien señales permaneciendo, no obstante, eléctricamente aislados; las "ondas" de voltaje no dañan, en consecuencia, al chip.

La MIDI difiere del diseño de la interface RS232 en un importante sentido. La velocidad de transmi-



Máquina de ritmos ("drum machine")

Glorioso Gliss
El *glissando* (deslizar el sonido de una nota a otra) se realiza fácilmente en un instrumento de cuerdas dejando resbalar los dedos hacia arriba o hacia abajo del traste. Para producir esto como un efecto de "resbalamiento", y no como una serie de notas marcadas, un sintetizador necesita una cuidadosa programación

sión de datos con la RS232 es de 1 920 palabras en serie por segundo, o 19,2 Kbaudios; la MIDI es un 50 % más lenta. Ello no afecta la compatibilidad con ordenadores personales, porque el sistema de circuitos lógicos dentro de la propia MIDI establece una nueva velocidad de reloj de 3 125 palabras por segundo, o 31,25 Kbaudios. Esta velocidad es relativamente elevada para la transmisión en serie, pero no es lo suficientemente rápida. Más adelante, en este mismo capítulo, veremos por qué.

En un sistema de música, el ordenador puede jugar dos posibles papeles, según la unidad MIDI en uso: en primer lugar, antes que nada debe colocar a la MIDI en modalidad activa y proporcionar la base de memoria para grabar la música; en segundo lugar, ha de soportar el software MIDI si el mismo no está residente en la propia unidad. Del mismo modo, la MIDI puede ser simplemente una interface digital entre los instrumentos musicales y el ordenador, o puede ser una interface con control activo sobre los datos transmitidos.
Existen dos modalidades de transmisión: grabación (*record*) y reproducción (*playback*). En la primera, la música producida con los instrumentos se envía a través de la MIDI a la memoria (ya sea del ordenador o del buffer de la MIDI). En la segunda, esta información digital es procesada por la MIDI de camino hacia los instrumentos: la información de tiempo, sincronización y control se le acopla a la misma tal como fue especificada en el protocolo definido por el usuario

Kevin Jones

La MIDI está diseñada para la conexión en interface con más de un instrumento receptor. Cuando hay más de un instrumento recibiendo instrucciones de la MIDI, la primera exigencia debe ser que cada uno de los datos se envíe al instrumento apropiado; de lo contrario, una máquina de ritmos podría acabar intentando tocar una melodía cuidadosamente secuenciada, y un sintetizador polifónico reproduciendo un patrón de bombo en *do central*. Los instrumentos compatibles con la MIDI deben poseer un ID o código de identificación numérico. Al código se le asigna uno de los 16 canales disponibles de la MIDI, de modo que solamente ese canal acepte datos para ese instrumento. La primera parte de una transmisión MIDI completa es, entonces, un byte de estado que incluye el ID. Todos los datos que siguen a esta instrucción de ruta pueden entonces especificar cómo se debe interpretar la instrucción.

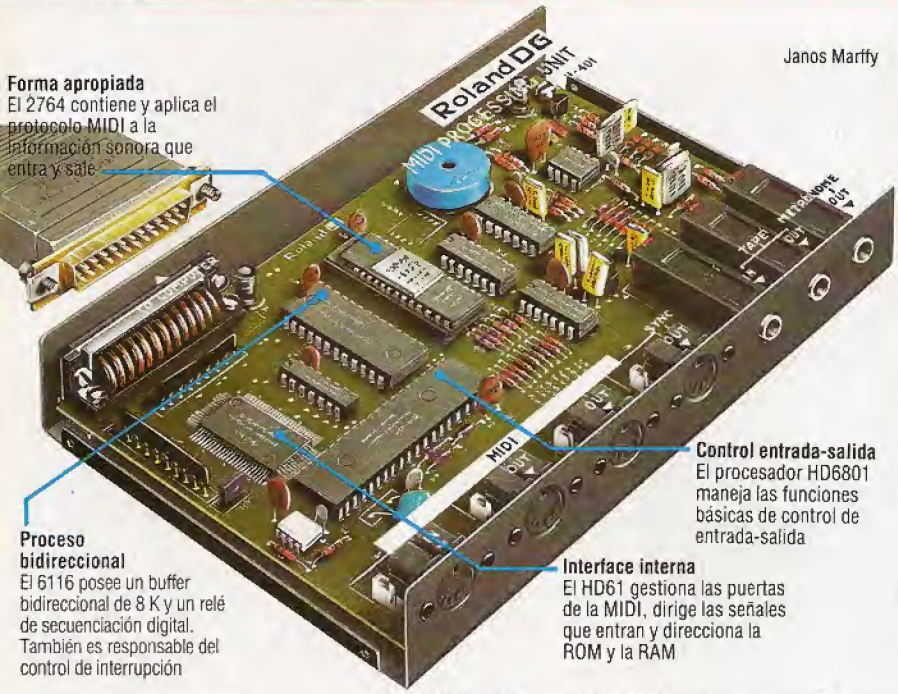
La unidad, con un tamaño aproximado de 100×120×45 mm, posee dos conectores DIN de cinco patillas, señalados como "MIDI IN" y "MIDI OUT". El primero acepta todas las instrucciones provenientes de un microordenador o sintetizador maestro, y "MIDI OUT" transmite al instrumento receptor la corriente de bits modificada. Muchos modelos poseen, asimismo, "MIDI THRU", un segundo conector de salida que simplemente transmite la corriente de bits original sin modificar enviada a "MIDI IN". Ésta se puede enviar entonces a una segunda interface. El cable, de una longitud máxima de 15 metros, está equipado con enchufes DIN de cinco patillas, y se conecta en el panel posterior de un microordenador o sintetizador maestro.

Do central

Pensemos en alguien que utiliza por primera vez una MIDI y desea probar una breve melodía. Ésta comienza en *do central*, sube a *mi*, después a *sol*, etc. La forma de indicar esto dependerá del tipo de software de música que esté utilizando. Tal vez esté empleando un lápiz óptico para ir marcando las notas en un pentagrama visualizado en la VDU. Este pentagrama se ha venido utilizando como formato de notación estándar en la música occidental desde hace más de cuatro siglos. Quizá esté entrando información en el teclado alfanumérico de su microordenador, empleando alguna clase de MCL (*Music Composition Language*: lenguaje para composición musical), nuevamente con una visualización VDU. Otra alternativa sería que estuviera tocando las notas de la melodía en un periférico de teclado musical. Este teclado podría no tener sonido propio, sino que produciría una u otra de las visualizaciones mencionadas anteriormente. Pero, cualquiera que sea la forma de entrar la música, la transmisión MIDI siempre será la misma: para empezar la melodía, el primer byte (transmitido como una palabra en serie con sus dos bits extras del chip ACIA) generará PLAY / ON CHANNEL 6; el segundo byte, MIDDLE C (tocar/en canal 6, *do central*, o do_3).

Esta mínima instrucción producirá la nota *do central* del instrumento receptor. Y el sintetizador continuará tocando *do central* a menos que también haya una instrucción que limite su duración, como STOP PLAYING / ON CHANNEL 6 (dejar de tocar en canal 6), byte uno; MIDDLE C *(do central)*, byte dos;

Janos Marffy



Forma apropiada
El 2764 contiene y aplica el protocolo MIDI a la información sonora que entra y sale

Control entrada-salida
El procesador HD6801 maneja las funciones básicas de control de entrada-salida

Proceso bidireccional
El 6116 posee un buffer bidireccional de 8 K y un relé de secuenciación digital. También es responsable del control de interrupción

Interface interna
El HD61 gestiona las puertas de la MIDI, dirige las señales que entran y direcciona la ROM y la RAM

La interface digital para instrumentos musicales
La interface MIDI concilia los protocolos de entrada-salida del ordenador y de los instrumentos musicales a él conectados (exactamente igual que cualquier otra interface), posibilitando, por consiguiente, que los instrumentos utilicen la memoria del ordenador. También procesa el sonido digitalizado que pasa a través de ella, agregando información relativa a control, sincronización y tiempo a la entrada del sintetizador

y ALLOWING FOR A DURATION OF X (permitiendo una duración de X), byte tres. Si no se suministra esta instrucción, y el resto de la melodía, *mi, sol*, etc., se entra sin parámetros de duración, todas las notas seguirán sonando. El resultado será un acorde de mantenido compuesto por las notas entradas.

Afortunadamente, incluso un somero conocimiento de la notación de pentagrama en la VDU será suficiente para indicar que lo que se pretendía que fuera una melodía es probable que se convierta en un acorde. Y un usuario de MCL que haya incurrido en tal error podrá simplemente ejecutar la secuencia, pero esta vez enviando una instrucción MONOPHONIC (monofónico) al sintetizador. *Monofonía* significa sencillamente un sonido, por oposición a muchos (*polifonía*). Si el sintetizador receptor puede producir sólo un sonido por vez cuando se establece en MONOPHONIC, la secuencia, pobremente introducida, se puede ejecutar como una sucesión de notas individuales solamente; una melodía, más que un acorde.

Supongamos que, al cabo de algunos ensayos y errores, el usuario novel ha entrado su frase musical correctamente, y que el sintetizador en interface esté ahora tocando. La melodía conserva el compás, y el ritmo (definido por la secuencia de duraciones) es correcto. Hay que destacar que hasta el momento de nuestro análisis los tipos de instrucción han sido bastante limitados. Sólo hemos hecho alusión a dos parámetros o características musicales: altura (*do central, mi, sol*, etc.) y duración.

El compositor de la melodía la escucha varias veces, y le parece que suena un poco "dura" (como es probable que suene mientras sólo posea un mínimo de definición). Opta, entonces, no por que del primer *do* se pase al *mi* secamente, sino por que desde el *do* vaya subiendo el tono gradualmente hasta el *mi*. Esta clase de movimiento se denomina *glissando* o deslizamiento de tono, y será característico de la forma en que una persona entone la melodía. En este contexto se dará a la actuación del sintetizador un toque de agilidad. De modo que ahora esta instrucción sustituye a la original para *do central*, agregando un byte extra.

Esto nos lleva a un punto muy simple que concierne a la interface MIDI. Si el sintetizador receptor no posee facilidad para producir *glissandi*, no puede ejecutar esta última instrucción. Podría tocar el *do central* tal como si estuviera recibiendo la instrucción *original*, o bien hacer algo completamente diferente. Si las instrucciones del usuario de una MIDI han de producir una sección de música polifónica y el sintetizador receptor es un instrumento sólo monofónico, es probable que éste haga una selección impredecible de la polifonía y luego la toque de forma monofónica. En resumidas cuentas, utilizar la MIDI para conectar un microordenador con un sintetizador muy básico no convertirá a éste en un sintetizador caro como el Fairlight.

Estas limitaciones también se aplican en sentido inverso. Puede que el instrumento receptor sea un soberbio y caro sintetizador, pero a menos que se definan suficientes parámetros musicales y que se establezcan de la forma deseada los propios controles del sintetizador, el resultado bien podría sonar con la musicalidad de una calculadora de bolsillo.

En la práctica, la segunda de estas dos situaciones es muy fácil de mejorar. Se deben establecer como *constantes* la mayor cantidad posible de parámetros utilizando los controles del sintetizador, y las instrucciones de la MIDI deben trabajar dentro con esos parámetros. Lo más probable es que sea éste el enfoque que adopte el músico de sintetizador cuyos problemas hemos considerado antes.

Hasta este momento hemos analizado las características de altura y duración, pero la MIDI puede estar provista hasta de 128 controles teóricos, que cubren filtración, distorsión, "ruido blanco" (todas las frecuencias posibles) y "ruido rosa" (frecuencias de escala media), cada uno de ellos con valores comprendidos entre 0 y 128. Esto es más que adecuado para tratar con los parámetros de que disponen la mayor parte de los sintetizadores, y son estos controles los que probablemente les interesen a los usuarios de microordenadores.

Y es aquí donde la velocidad de transmisión de la MIDI adquiere relevancia. Hemos visto que una instrucción muy directa, relativa a una única nota y definiendo sólo dos parámetros, utilizaba tres palabras en serie. Con la velocidad de 31,25 Kbaudios, esto lleva casi un milisegundo. Los acordes de seis notas son comunes en muchos tipos de música: transmitir tal acorde ocuparía 5,76 milisegundos. Si ahora comenzamos a definir aún más este acorde utilizando controles MIDI, el tiempo de transmisión se vuelve demasiado lento y el oído humano comienza a detectar cambios en las características del sonido, producidos por la demora. Estos cambios se hacen evidentes sólo cuando los sonidos, en especial los similares, se producen juntos; pero como interface de audio, la MIDI se diseñó para manipular sonidos simultáneos. La música, lamentablemente, es un medio "en paralelo": como oyentes, estamos habituados a oír que las cosas sucedan de forma simultánea.

Por consiguiente, no es sorprendente que la transmisión en serie de la MIDI haya sido objeto de críticas; la transmisión en paralelo habría realizado mejor la tarea. Queda por ver si a los usuarios de la MIDI este fallo les molesta. En el presente, el diseño de la interface parece mantener un "delicado equilibrio" entre costo y eficacia, de modo que vale la pena tener presente que las especificaciones actuales bien podrían ser sólo las primeras.

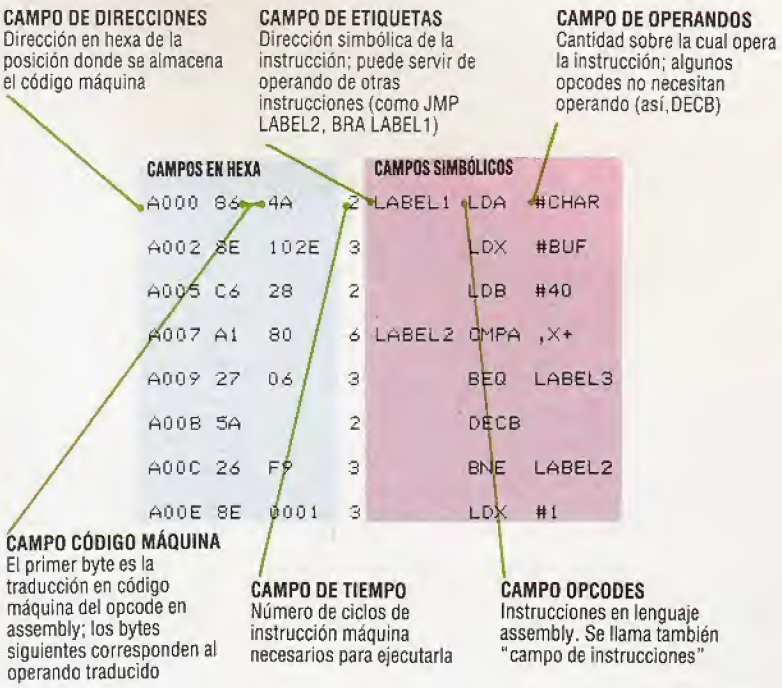
Cuestión de registros

Vamos a estudiar los registros del 6809 con más detalle, analizando su funcionamiento y algunas de las instrucciones básicas que suelen utilizar

Ya hemos visto que un registro es una posición de memoria dentro del mismo chip del procesador, y observamos cómo la programación en assembly implica el manejo de valores almacenados en los registros y su traslado de o a la memoria. Vimos posteriormente que algunos registros (en especial los que almacenan y procesan direcciones) son de 16 bits, mientras que otros son de ocho bits, y que cada registro realiza una función distinta.

- Los **registros índice** sirven para modificar las direcciones empleadas en nuestro programa.
- Los **índices de pila** son usados por el procesador como zona de trabajo y el programador los puede utilizar para recuperaciones y almacenamientos rápidos.
- El **contador del programa** retiene la dirección de la instrucción siguiente a ejecutar, pudiéndose alterar el flujo secuencial del programa modificando su contenido.
- Los **acumuladores** son los registros más frecuentemente utilizados, y se emplean para realizar funciones aritméticas.
- El **registro de código de condición** contiene varios flags que representan el estado del procesador (por ej., si es cero el resultado de una operación). Estos flags pueden ser consultados con el fin de hacer una selección o un bucle, proporcionando al assembly una estructura IF...THEN.

El procesador 6809 contiene todos estos registros. Puesto que se trata de una variante mejorada del primitivo procesador Motorola 6800 (al igual que el 6502, empleado por el BBC Micro y muchos otros), existen muchas similitudes entre los lenguajes assembly de ambos procesadores. Sin embargo no son compatibles y la codificación escrita para el uno no podrá ser ejecutada por el otro. Muchos programas del 6800 tienen el código fuente compatible: un programa en assembly escrito para el 6800 puede ser reensamblado para el 6809 con buenas posibilidades de que funcione. Pero ni siquiera este pequeño peldaño de compatibilidades está al alcance del 6502 (o de su posterior desarrollo, el 6510 que emplea el Commodore 64). De todos modos, las similitudes entre ambos procesadores hacen al menos que la tarea traductora de un programa en versión del 6809 no sea demasiado difícil, y puede servir de buena introducción al 6809 para el que ya esté familiarizado con el 6502. Los registros que contiene el 6809 son:



Campos de estudio
Los programas en lenguaje assembly se parecen muy poco a los listados de programas en BASIC, y una vez traducidos a código máquina mediante un programa ensamblador todavía aparecen más desconcertantes. La clave para su comprensión es fijarse en las columnas (o campos) adecuadas (por lo general, los campos de etiquetas, de operandos y de opcodes), ignorando el resto. Esta muestra le puede ayudar

- Dos acumuladores de ocho bits, conocidos como A y B. No existe diferencia funcional alguna entre ellos, pudiéndose usar cualquiera de los dos, cuando son empleados como registros de ocho bits. Pero el hecho de que sean dos permite retener valores en uno de ellos mientras se trabaja con el otro. O también pueden usarse conjuntamente como si fuera un único acumulador de 16 bits, facilidad poderosa que permite realizar operaciones aritméticas directamente en 16 bits. Precisamente por este motivo se dice que el 6809 es un pseudoprocesador de 16 bits, aunque esto no es así, sino más bien un avance dentro de la línea de los procesadores de ocho bits. Si se emplean juntos A y B, son llamados registro D de 16 bits.
- Dos registros índice, el X y el Y. De nuevo nos encontramos con que su funcionamiento es idéntico, siendo indiferente el empleo de uno u otro. Sin embargo, existe una ligera diferencia operativa, ya que algunas instrucciones que se sirven de Y se traducirán como instrucciones de dos bytes, mientras que las correspondientes al registro índice X serían de un solo byte, haciendo de esta manera el programa un poco más largo y lento. Siempre que se necesite solamente un registro, es preferible utilizar X.
- Dos índices de pila, S y U. El procesador emplea S para todas sus operaciones con la pila. Por eso, el programador, aunque puede usar indistintamente S o U, deberá asegurarse de que empleando S no afectará la operación del procesador. Es, pues,

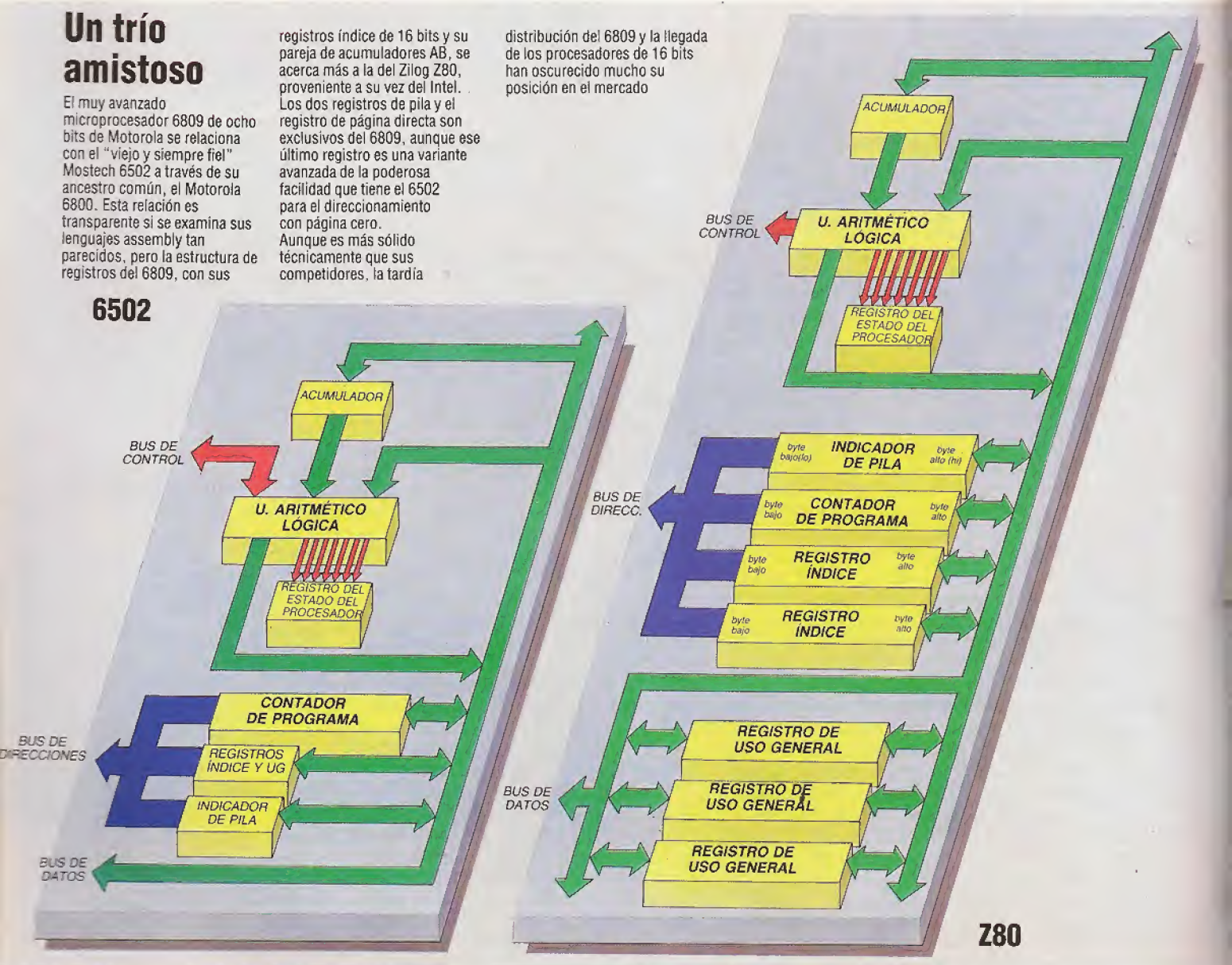
más seguro el empleo de U. Estos dos registros de pila hacen del 6809 un procesador ideal con FORTH.

- La utilización de los registros X, Y, S y U no tiene por qué limitarse a sus destinos originales: tanto S como U pueden servir de registros índice, por ejemplo, y los cuatro registros pueden emplearse para almacenar y manipular números de 16 bits.
- El contador de programa (PC) es un registro de 16 bits que se ajusta automáticamente señalando la instrucción siguiente a ejecutar. Las instrucciones JMP (salto) y BRA (bifurcación) alteran su contenido y el 6809 permite que sea usado como una especie de registro índice. Ya analizaremos más adelante los posibles efectos; aquí basta saber que la utilización de direcciones relativas al contenido del PC, en vez de direcciones absolutas almacenadas en X o Y, brinda al programador un código independiente de la posición. En otras palabras, los programas escritos adecuadamente pueden cargarse en cualquier posición de la memoria y ser ejecutados desde allí.
- El registro de código de condición (CC) tiene ocho bits, empleados por separado como flags indicativos de la condición del procesador. Una instrucción como BNE (*branch not equal*: bifurcar si no es igual) consulta uno de estos flags, causando un cambio en el flujo del control según la condición del flag (cero o uno).

Existen diversas instrucciones destinadas al traslado de los datos dentro, fuera o entre los diversos registros. Sirvan los siguientes ejemplos. Supongamos que tenemos reservadas algunas posiciones de memoria mediante las directivas de assembly FCB y FDB. El procesador no las toma como instrucciones. En el momento de traducir el programa assembly a código máquina, las obedecerá inmediatamente reservando las posiciones deseadas para uso del programa. El ensamblador no traducirá las directivas a código máquina porque no tienen interés para el procesador. Dado que se parecen a códigos de operación en assembly (del tipo BNE o JMP) se suelen llamar *pseudo-ops*; pero es preferible el nombre de *directivas de assembler*, que explica mejor su función: dirigir el funcionamiento del programa assembler. En el caso de que tales directivas estén etique-

Un trío amistoso

El muy avanzado microprocesador 6809 de ocho bits de Motorola se relaciona con el "viejo y siempre fiel" Mostech 6502 a través de su ancestro común, el Motorola 6800. Esta relación es transparente si se examina sus lenguajes assembly tan parecidos, pero la estructura de registros del 6809, con sus registros índice de 16 bits y su pareja de acumuladores AB, se acerca más a la del Zilog Z80, proveniente a su vez del Intel. Los dos registros de pila y el registro de página directa son exclusivos del 6809, aunque ese último registro es una variante avanzada de la poderosa facilidad que tiene el 6502 para el direccionamiento con página cero. Aunque es más sólido técnicamente que sus competidores, la tardía distribución del 6809 y la llegada de los procesadores de 16 bits han oscurecido mucho su posición en el mercado

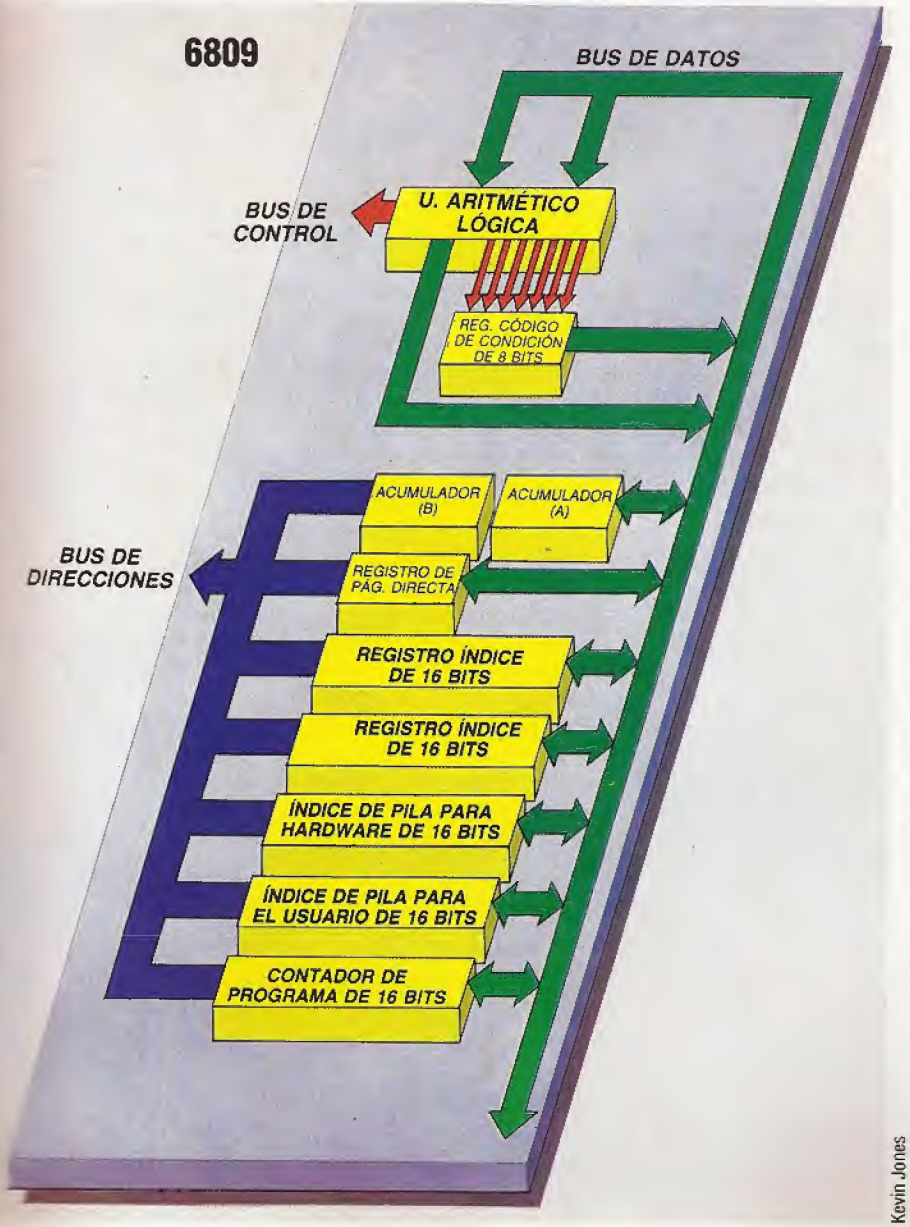


tadas, el assembler traducirá la etiqueta en la dirección apropiada, como en este ejemplo:

```
NUM1 FCB 0      reservar un byte que se denominará NUM1, con valor inicial 0
NUM2 FCB 0      similar al anterior pero etiquetado con otro nombre
NUM3 FDB #A93B  reserva dos bytes para el número de 16 bits #A93B (el símbolo # se utiliza para indicar que un número está escrito en notación hexadecimal)
```

Las siguientes instrucciones tienen por finalidad cargar los valores almacenados en tales posiciones dentro de los diversos registros (LD; del inglés *load*: cargar):

```
LDA NUM1   cargará en el acumulador el número de ocho bits almacenado en la posición de memoria referenciada por NUM1
LDB NUM2   carga, como antes, en el acumulador B el contenido de NUM2
```



Kevin Jones

```
LDX NUM3
LDY NUM3
LDS NUM3
LDU NUM3
LDD NUM3   instrucciones que cargarán el número de 16 bits contenido en NUM3 dentro de los registros X,Y,S,U y D respectivamente
```

De forma similar, el contenido, de 8 o 16 bits, de cualquier registro puede almacenarse en una posición de memoria por medio de una de estas instrucciones (ST; del inglés *store*: almacenar):

```
STA NUM1
STB NUM2
STX NUM3
STY NUM3
STS NUM3
STU NUM3
STD NUM3
```

Adviértase que cuando el acumulador es cargado (LD) con el valor de NUM1 el contenido de la posición de memoria NUM1 no se altera; de forma similar ocurre con las operaciones de almacenamiento (ST).

Se pueden intercambiar *(exchange)* contenidos de dos registros (siempre que ambos sean del mismo tamaño) mediante la instrucción EXG. Por ejemplo:

```
EXG A,B  lo que contiene A se traslada a B y viceversa
EXG X,S  intercambio de contenidos entre X y S
```

Lo que contiene un registro se puede trasladar *(transfer)* a otro registro. Por ejemplo: TFR Y,U copia en U lo que contiene Y. De nuevo es condición indispensable que ambos registros sean del mismo tamaño (8 a 16 bits).

Para terminar esta lección, vamos a presentar una instrucción que realiza una tarea concreta y que nos permitirá escribir un sencillo programa. La instrucción ADD (en inglés, "sumar") le suma al valor contenido en el acumulador un valor contenido en una posición de memoria (recuérdese que disponemos de dos acumuladores):

```
ADDA NUM1  significa: "suma el contenido de la posición NUM1 al contenido del registro A y el resultado déjalo en dicho registro A"
```

Vamos primero a sumar los dos números de ocho bits que están en NUM1 y NUM2, dejando el resultado en NUM1 y descuidando la posibilidad de un desbordamiento si la suma supera un número de ocho bits. Sumaremos después los contenidos de las dos posiciones, pero con la obtención de un resultado de 16 bits que se dejará en NUM3.
Primer ejemplo:

```
LDA NUM1   carga el primer número dentro de A
ADDA NUM2  agrega a A el segundo número
STA NUM1   almacena el resultado en NUM1
```

Segundo ejemplo:

```
LDB NUM1   carga el primer número en B
SEX        convierte el número anterior en un número de 16 bits cargándolo en D
STD NUM3   almacena D en NUM3
LDB NUM2   carga el segundo número en B
SEX        conversión a 16 bits y carga en D
ADDD NUM3  suma a D el contenido de NUM3, que es un número de 16 bits
STD NUM3   almacena el resultado en NUM3
```

Cuidando la imagen

La creatividad australiana se halla detrás de la alta calidad del software producido por la empresa Melbourne House

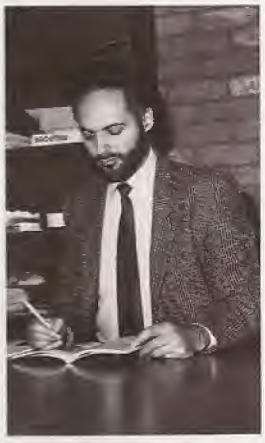
Melbourne House fue fundada en 1977 por el australiano Fred Milgrom. El lanzamiento del Sinclair ZX80 alertó a Milgrom de los potenciales beneficios de editar libros sobre la informática personal, y en 1980 Melbourne House produjo *30 Programs for the ZX80 (Treinta programas para el ZX80)*. El éxito de esta publicación fue el origen de una serie de libros dedicados a la máquina Sinclair, y la empresa sacó al mercado su primer paquete de software: *Space invaders (Invasores del espacio)*, nuevamente para el ZX80.

El año siguiente vio la aparición del ZX81. La demanda de libros y cassettes para el ZX80 cayeron a plomo, y fueron las ventas en Estados Unidos las que salvaron a la empresa del desastre. La lección se aprendió bien y Melbourne House comprendió las ventajas de la diversificación. A medida que iban saliendo al mercado nuevas máquinas, la empresa surtía a los usuarios de libros y software, cuyas ventas se veían favorecidas por la baja calidad de las guías para el usuario que se suministraban con algunos ordenadores personales.

El éxito inmediato del Sinclair Spectrum facilitó que Melbourne House produjera software de juegos explotando los gráficos en color en alta resolución y el sonido de la máquina. El juego recreativo *Penetrator* se vendió bien, pero el mayor éxito de la empresa fue el juego *The Hobbit*, una aventura gráfica basada en la novela homónima de J.R.R. Tolkien, que ganó el premio Golden Joystick al mejor juego de estrategia del año. La cassette del juego se comercializó en un paquete que también contenía un ejemplar del libro de Tolkien; ésta fue una condición que impusieron los albaceas de los bienes del autor y determinó que *The Hobbit* se vendiera a un precio tres veces superior al de la mayoría de los paquetes de software que se editaban para el Spectrum en aquellos tiempos. A pesar del costo, las ventas fueron sumamente buenas y ahora *The Hobbit* se comercializa también para otras máquinas, incluyendo el BBC Micro y el Oric/Atmos.

Los programadores de la empresa están afincados en Melbourne (Australia). Cada equipo, compuesto por cuatro miembros, se concentra en un aspecto de un juego. Ello significa que se invierte más tiempo en el desarrollo de los juegos, pero esta política les ha valido enormes ventas y la lealtad de los clientes. La empresa espera capitalizar esta fidelidad en una campaña para vender más libros. Paula Byrne, directora de publicidad de la empresa, señala: "Cuando la gente va a comprar un libro no sabe bien lo que desea y, por tanto, es probable que acabe adquiriendo algo que no es adecuado, lo que les hace perder el interés por adquirir otros libros". Melbourne House confía en combatir esta confusión clasificando sus libros como para lectores principiantes, intermedios o avanzados, y empaquetando libros y software en el mismo estilo, de modo que el público identifique libros de calidad con software de calidad. Con este fin, la empresa incluye en cada uno de sus productos una ficha de registro en la que se invita a los clientes a dar una opinión acerca de la calidad de su compra.

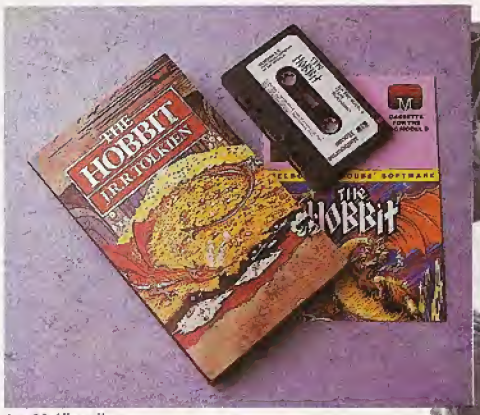
De su último juego, *Mugsy*, la empresa afirma que es "la primera tira cómica interactiva para ordenador del mundo", y permite que el jugador asuma el control de una banda de maleantes en el Chicago de los años veinte. Los gráficos son muy detallados y responden a un diseño fantástico, si bien el juego propiamente dicho es poco complicado. Melbourne House está trabajando, asimismo, en una aventura "estilo *Hobbit*" denominada *Sherlock Holmes*, que saldrá a la venta dentro de poco. Se dice que este juego es tan innovador como lo fue *The Hobbit* en la época de su lanzamiento, y su desarrollo ha llevado 15 meses. Es muy poco lo que ha trascendido acerca del contenido del juego, ¡aunque se dice que requiere un buen conocimiento del transporte victoriano!



Alfred Milgrom, codirector y editor de Melbourne House

Philip Mitchell, autor del juego "The Hobbit" y de "Sherlock Holmes"

El libro del juego
Con *The Hobbit*, además de crear un juego divertido y emocionante, Melbourne House también incluye un ejemplar de la obra maestra de J.R.R. Tolkien; esta iniciativa ha constituido una brillante táctica de ventas



Ian McKinnell

Equipo de software de Melbourne House

Especialidad japonesa

Los ordenadores de bolsillo se utilizan para diversas aplicaciones serias. Esta vez comparamos varias máquinas de la gama Casio

Las calculadoras programables se pueden utilizar en muchas aplicaciones similares, pero los ordenadores de bolsillo tienen la ventaja definitiva de que pueden emplear BASIC y manipular texto, mientras que las calculadoras programables utilizan sus propios lenguajes especiales (que con frecuencia se asemejan al lenguaje máquina) y son aptas sólo para tareas numéricas. Los ordenadores de bolsillo son ideales para aquellas actividades en las que es necesario efectuar cálculos que implican una fórmula establecida. Tales cálculos suelen ser repetitivos y tediosos; los usuarios de ordenadores de bolsillo pueden escribir sus propios programas para tratar este tipo de cálculos.

De la gama de ordenadores de bolsillo Casio, bien establecida, el más barato es el FX-720P. Sus dimensiones son de 165×85×15 mm y pesa sólo 210 g. El FX-720P posee un teclado cuyas dimensiones representan algo más de la mitad de las de un teclado normal. Utiliza el trazado QWERTY convencional, con la excepción de que todas las filas están alineadas en vez de seguir la disposición alternada habitual.

Cada tecla alfabética puede producir otros dos caracteres (como signos de puntuación) y palabras clave de BASIC, cuando se utilizan junto con las teclas Shift o Function. Sin embargo, a diferencia de otros teclados, estas teclas no se emplean simultáneamente con la tecla alfabética (ALPHA); se pulsa primero una y se suelta antes de pulsar la tecla apropiada. Ello hace que resulte fácil trabajar en el teclado con una sola mano (dejando la otra libre para sostener el ordenador).

El FX-720P utiliza una visualización en cristal líquido (LCD) con 12 caracteres en una única línea, y las líneas más largas se visualizan pulsando dos teclas del cursor para desplazamiento a izquierda o derecha. Moviendo con el pulgar una ruedecilla que hay a uno de los lados de la visualización se ajusta el contraste de la LCD de modo que se adapte a los distintos ángulos de vista y niveles de iluminación. Si el ordenador no se utiliza durante seis minutos, la visualización se apaga para ahorrar energía.

El FX-720P se vende con una escasa memoria de dos Kbytes. No obstante, toda la memoria para el usuario se proporciona en forma de cartuchos enchufables que conservan su contenido cuando se quitan de la máquina. Un usuario puede, en consecuencia, almacenar distintos programas y datos en cartuchos separados, que se enchufarán cuando sea necesario.

Dado que los cartuchos pueden retener tan pocos datos, se llenan enseguida, debido especialmente a que el ordenador permite retener en la memoria hasta 10 programas en BASIC al mismo tiempo. Se debe seleccionar una modalidad especial para dar entrada a un programa en BASIC y se necesita otra para ejecutarlo. La versión de BASIC del FX-720P es sorprendentemente buena para tratarse de una máquina tan pequeña: incluye casi todas las instrucciones y funciones estándares, a excepción de CHR$ y ASC. Una instrucción simple, BEEP, permite producir dos sonidos diferentes.

Se considera que el FX-720P se halla en el extremo inferior de la gama Casio de ordenadores personales y, por consiguiente, la mayoría de sus funciones se ven superadas en los otros modelos. Existe, no obstante, una característica que es exclusiva de esta máquina. Se trata del *Data bank* (banco de datos), que es un pequeño programa de base de datos. Utilizando este programa se puede entrar fácilmente información del tipo nombres y direcciones, gastos o citas. El programa buscará cualquier ítem de información e incluso permite diferentes niveles de búsqueda; por ejemplo, si con una búsqueda se ha encontrado una gran cantidad de ítems, se puede realizar otra búsqueda en estos mismos utilizando otro criterio. Esta característica hace que la máquina sea una competidora directa del Psion Organiser, aunque el FX-720P cuesta menos y es más versátil.

Casio FX-750P y PB-700

Si el FX-720P parece estar dirigido a los hombres de negocios, el siguiente modelo de la gama Casio, el FX-750P, pretende cubrir las necesidades de científicos e ingenieros. Ofrece una visualización de

Modelo económico
El ordenador personal FX-720P de Casio tiene 2 K de memoria en cartuchos enchufables, una visualización en cristal líquido de una línea y un juego completo de teclas alfabéticas y numéricas. Se pueden enchufar cartuchos adicionales para proporcionar memoria extra

Chris Stevens

Interface FA-20
Cada uno de los ordenadores de bolsillo de la gama Casio se puede acoplar a una interface de este tipo. Este modelo permite albergar limpiamente el FX-750P. La FA-20 viene con pilas recargables y un adaptador de corriente

Chris Stevens

24 caracteres y un teclado ligeramente mejorado. Viene con un cartucho de memoria de dos Kbytes, del mismo tipo que utiliza el FX-720P, pero además posee una ranura para un segundo cartucho. Esto significa que su capacidad de memoria se puede ampliar hasta ocho Kbytes. La máquina mide 185 mm de ancho y pesa apenas 250 g. A diferencia del FX-720P, las teclas Shift y Function se deben utilizar al mismo tiempo que la tecla alfabética.

En diez de las teclas hay programadas importantes constantes: la velocidad de la luz, la aceleración gravitatoria de la Tierra, la constante de Planck, la constante de Boltzmann, la constante de Avogadro, la carga elemental, la masa atómica, la masa electrónica, la constante gravitatoria y el volumen molar. Los físicos y los químicos utilizan con mucha frecuencia estos números en sus cálculos y el hecho de tenerlos fijos en la memoria les evita el problema de tener que recordarlos y digitarlos. El ordenador también incluye funciones de las que el BASIC normalmente no dispone: seis funciones hiperbólicas y un juego de funciones estadísticas que incluyen desviación estándar, regresión lineal y correlación. El BASIC está mejorado en otros varios sentidos de modo tal que no sería inapropiado para muchos ordenadores personales.

El tercer ordenador de bolsillo de Casio es el recientemente lanzado PB-700. Posee una visualización de 20 caracteres por cuatro líneas y cuatro Kbytes de memoria. La memoria se puede ampliar hasta 16 Kbytes utilizando paquetes de memoria que se enchufan en el lado de abajo del ordenador. Tiene casi 200 mm de ancho y pesa 315 g. Al igual que los otros ordenadores de bolsillo, viene con una carcasa blanda de protección.

El PB-700 tiene un teclado semejante al del FX-750P, con la excepción de que tiene una tecla Caps en lugar de la tecla Function. Mientras se mantiene pulsada esta tecla, el ordenador produce letras en minúscula. La versión de BASIC de la máquina es similar a la versión del FX-750P, si bien posee algunas instrucciones extras para dibujar gráficos en la LCD, que tiene una resolución de 32 por 160 puntos.

El BASIC incluye, asimismo, instrucciones para trazar gráficos utilizando la unidad interface opcional de impresora-plotter en color y cassette, denominada FA-10. Ésta se acopla al ordenador. Utiliza cuatro bolígrafos de distintos colores para dibujar texto y gráficos en un papel de 115 mm de ancho. Las instrucciones del BASIC permiten que la impresora-plotter dibuje líneas, círculos y ejes para gráficos.

La FA-10 también lleva incorporada una interface de cassette que le permite utilizar una grabadora de cassette normal para guardar programas y datos. Considerando lo pequeña que es la memoria del ordenador, ésta es una facilidad muy útil. Al instalar en la impresora-plotter una diminuta grabadora de cassette se obtiene un sistema informático completo lo suficientemente pequeño como para poderlo llevar a cualquier parte. También hay a la venta una interface Centronics separada que permite utilizar con el PB-700 las impresoras para ordenador estándares.

Para el FX-720P y el FX-750P también hay impresoras e interfaces de cassette. El FX-720P utiliza la impresora FP-12S. Una unidad separada, la FA-3, actúa como interface para una grabadora de cassette. El FX-750P utiliza la FA-20, una diminuta impresora e interface de cassette combinadas. Tiene un espacio para almacenar un cartucho de memoria y viene con pilas recargables, un transformador de corriente y una carcasa protectora.

Los tres ordenadores se suministran con manuales exhaustivos. Éstos contienen todo lo necesario; pero es obvio que están traducidos del japonés y en muchos puntos la redacción resulta un tanto extraña. No es que sean inutilizables, sino que pueden resultar confusos para el principiante. Otro problema es que para estos ordenadores casi no hay software. Los usuarios tendrán que escribirse sus propios programas o bien copiar los programas de muestra que vienen listados en el manual de cada máquina.

PRODUCTO	DIMENSIONES	PESO	MEMORIA	VISUAL.	OBSERVACIONES
FX-720P	165×85×15 mm	210 g	2 K	1×12 car.	Base de datos incorporada. Teclado numérico
FX-750P	185×85×15 mm	250 g	2 K	1×24 car.	Ranura para segundo cartucho. Constantes físicas ya programadas. BASIC ampliado
PB-700	200×85×15 mm	315 g	4 K	4×20 car.	Capacidad para gráficos. Impresora-plotter

Tortuga juguetona

Vamos a analizar aquí una modalidad alternativa a la inmediata: la de edición, que permite crear nuevas instrucciones LOGO

Tal vez los *procedimientos* no les resulten familiares a muchos programadores de micros, pero el concepto de procedimiento está presente en "procedimientos de instrucciones" cotidianos tales como recetas de cocina y patrones de labores de punto. En este capítulo vamos a crear un procedimiento en LOGO al cual llamaremos CUADRADO. (A diferencia de las instrucciones de LOGO, que siempre se deben digitar en mayúsculas, los nombres de los procedimientos se pueden digitar en mayúsculas o bien en minúsculas.) Comenzamos por digitar:

```
EDIT CUADRADO
```

Se limpiará la pantalla y luego aparecerá el mensaje TO CUADRADO en la parte superior de la visualización a modo de recordatorio del nombre del procedimiento, y en la parte inferior aparecerá EDIT CTRL-C TO DEFINE CTRL-G TO ABORT (EDIT CTRL-C para definir CTRL-G para abortar). Este mensaje un tanto críptico sirve de ayuda para ir guiando sus acciones mientras esté en modalidad de "edición". EDIT (edición) simplemente le informa que el LOGO ya no está en modalidad inmediata sino que espera a que se cree un procedimiento. Una vez hecho esto y estando ya listo para almacenar su procedimiento, pulsando Control-C se definirá (registrará) el listado, mientras que Control-G permitirá abortar el procedimiento y volver a empezar.

En la modalidad de edición se puede digitar todo lo que se desee, pero en la modalidad no se obedecerá ninguna de las instrucciones. El LOGO simplemente toma nota de sus instrucciones y las almacena en su "diccionario" bajo el nombre que se le haya dado al procedimiento. Complete la definición del procedimiento CUADRADO entrando las siguientes instrucciones:

```
TO CUADRADO
  REPEAT 4 [FD 50 RT 90]
END
```

Ahora puede decirle al LOGO que defina este procedimiento (es decir, que lo recuerde) digitando Control-C; entonces recibirá el mensaje CUADRADO DEFINED (definido). Si ha cometido un error en su definición, puede abortar todo el proceso digitando Control-G, en cuyo caso debe volver a empezar, o bien puede utilizar el editor de pantalla completa de la modalidad de edición para efectuar correcciones. Éste permite usar las teclas del cursor para desplazarse hasta cualquier punto del procedimiento e insertar o eliminar caracteres. (El editor de línea de la modalidad inmediata permite efectuar correcciones sólo en la última línea entrada.) El LOGO posee muchas más instrucciones que hacen que la edición de procedimientos extensos resulte simple; nos ocuparemos de ellas más adelante.

Ahora que ha definido con todo éxito su procedimiento, veamos cómo funciona. Entre DRAW para acceder a la pantalla de gráficos, luego entre CUADRADO; ahora se obedecerán las instrucciones de la definición del procedimiento y la tortuga dibujará el cuadrado. Como puede ver, los procedimientos se utilizan exactamente de la misma manera que las instrucciones básicas del LOGO; sencillamente se entra el nombre del procedimiento y se ejecutan las instrucciones previamente definidas. Las instrucciones originales que están presentes cuando se carga el LOGO se denominan *primitivas*. Una vez que se le ha "enseñado" al lenguaje un nuevo procedimiento, éste se puede emplear exactamente de la misma forma que una primitiva. En otras palabras, el LOGO es un lenguaje "extensible", de modo que puede ser confeccionado a medida para que se adapte a las propias necesidades.

Si su procedimiento no hace lo que esperaba que hiciera, es muy fácil modificar la definición. Tal como lo hemos definido, CUADRADO dibuja un cuadrado con lados de 50 unidades de largo. Vamos a suponer que se prefiera un cuadrado más pequeño, supongamos que de 30 unidades de lado. Retornemos al editor digitando:

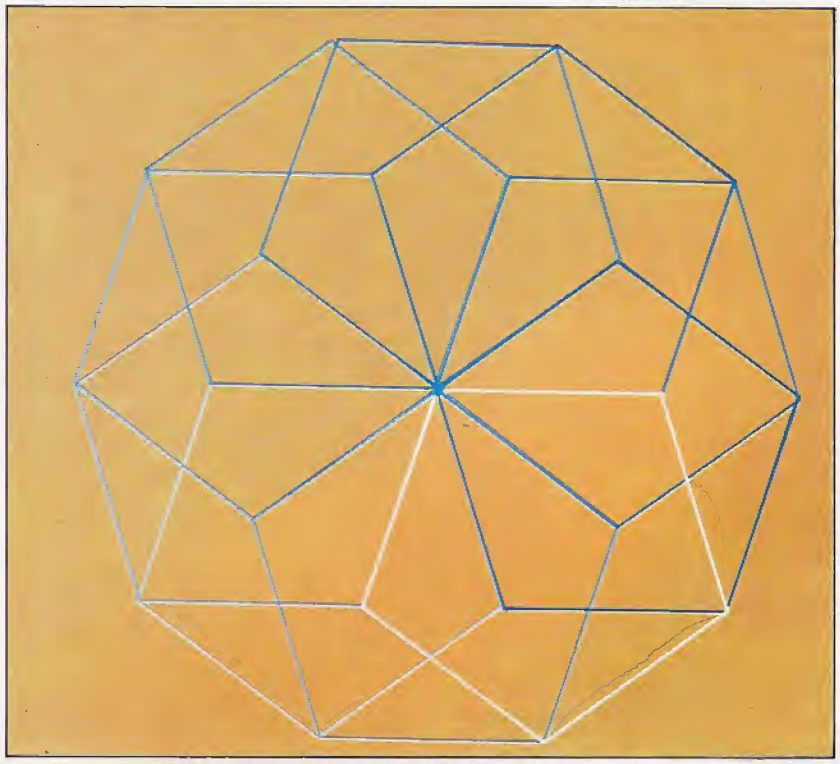
```
EDIT CUADRADO
```

Ahora se visualiza el texto de que consta el procedimiento y se puede utilizar el editor de pantalla para cambiar 50 por 30. Después de hecho esto, redefina el procedimiento mediante Control-C y digite CUADRA-

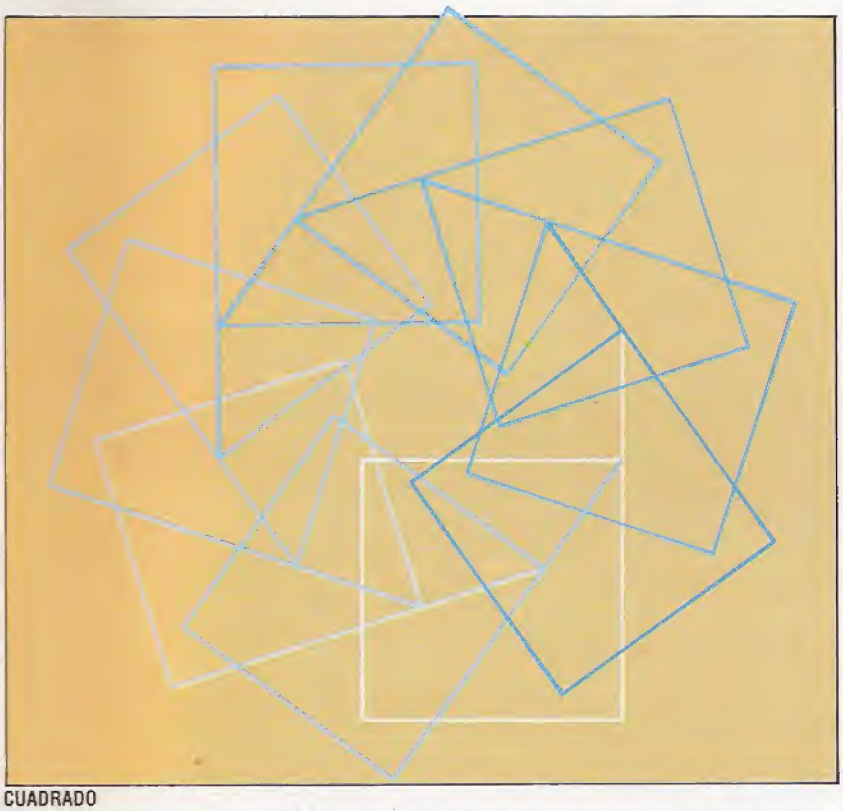
Abreviaturas

EDIT	ED
HIDETURTLE	HT
SHOWTURTLE	ST

PENTÁGONO



Liz Dixon



CUADRADO

DO para asegurarse de que ahora es del tamaño que se deseaba. A modo de ejercicio, intente escribir procedimientos para dibujar las formas que creó en el capítulo anterior (véase p. 1012): triángulos, rectángulos, pentágonos, estrellas, etc.

Ahora trate de emplear sus procedimientos con la instrucción REPEAT. Por ejemplo, defina PENT como:

```
REPEAT 5 [FD 50 RT 72]
```

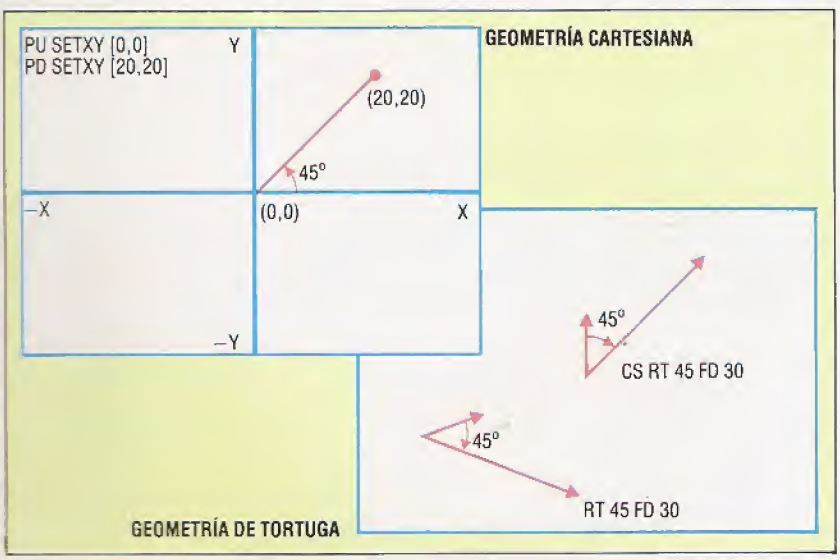
y después pruebe:

```
REPEAT 10 [PENT RT 36]
```

Puede experimentar con otras formas de esta misma manera, y pronto descubrirá que mediante la utilización de REPEAT se pueden construir formas complicadas rápida y fácilmente. Pruebe con ésta:

```
REPEAT 10 [CUADRADO RT 36 FD 25]
```

Cogito ergo LOGO
La tortuga se puede direccionar en términos de geometría cartesiana o de geometría "de tortuga". En la primera, las posiciones de la tortuga se expresan absolutamente: se miden desde los ejes X e Y imaginarios cuyo origen [0,0] es la posición CS; en geometría de tortuga, las instrucciones, relativas, se expresan en relación a la posición y el encabezamiento corrientes de la tortuga, sean cuales fueren



Luz Dixon

En el diagrama de la página contigua le mostraremos los resultados que da la tortuga. Los ejemplos que ofrecimos en el capítulo anterior estaban diseñados para que se experimentara con ellos; quizá sus intentos por dibujar las diversas formas sugeridas le hayan llevado a algunos resultados inesperados. Para dibujar un triángulo, tal vez primero haya probado con algo más o menos así:

```
REPEAT 3 [FD 50 RT 60]
```

¡y entonces habrá descubierto que ha creado la mitad de un hexágono! Si intentó dibujar la estrella de cinco puntas, ciertamente le habrá resultado mucho más difícil de lo que pensaba. Cuando uno trata con problemas de este tipo, suele ser de ayuda "jugar a la tortuga", imaginando que *uno* es la tortuga que se va moviendo por ahí. En realidad, se dice que es posible distinguir entre un programador de BASIC y un programador de LOGO ¡mirando cómo mueven los hombros mientras programan! Imagínese que dibuja cualquier forma cerrada desde el punto de vista de la tortuga. La tortuga ha de dar la vuelta completa alrededor de la forma y debe terminar en el mismo lugar donde empezó, mirando hacia la dirección original. De modo que debe girar a través de un múltiplo de 360°. Si la forma es "convexa" (si ninguno de sus ángulos interiores es mayor de 180°), entonces la tortuga habrá girado a través de 360° exactamente. En el caso de un triángulo, son tres los giros a efectuar, de modo que cada uno de ellos ha de ser de 360/3=120°. A partir de esto, se puede deducir que la instrucción correcta para dibujar un triángulo es:

```
REPEAT 3 [FD 50 RT 120]
```

Incluso para polígonos no convexos, como las formas en estrella, se aplica el mismo principio; la única diferencia es que en este caso el ángulo total descrito es un número entero múltiplo de 360° (porque se está creando más de una forma completa).

Estos principios son generales (no se aplican simplemente a los polígonos) y componen lo que se conoce como el *teorema del recorrido total de la tortuga*. Si examina la estrella de seis puntas desde el punto de vista de la tortuga, verá que ésta no puede dibujar la forma si sólo se mueve hacia adelante y gira describiendo el mismo ángulo cada vez; se requiere un procedimiento más complicado.

Geometría de las coordenadas de la tortuga

La geometría de tortuga trata de las propiedades de las formas en sí mismas y no de la relación de formas respecto a un punto de referencia exterior, como es el caso en la geometría de coordenadas. La geometría de tortuga es relativa: los movimientos se realizan en una cantidad especificada de unidades a partir de la posición en curso en la pantalla. La geometría de coordenadas utiliza valores absolutos: la pantalla se imagina como una cuadrícula, con una cantidad definida de unidades que se extienden vertical y horizontalmente respecto al centro. Cada punto de la cuadrícula tiene un valor numérico específico y el movimiento se define en relación a estas referencias. Sin embargo, se puede utilizar la geometría de coordenadas con la tortuga.

Por ejemplo, la instrucción SETXY 20 30 desplaza

a la tortuga desde su posición corriente hasta el punto (20,30). Si se ha especificado PENDOWN, la tortuga trazará una línea a medida que se vaya moviendo; por el contrario, la instrucción PENUP hará que la tortuga se mueva sin dejar ninguna huella. El origen (0,0) está en el centro de la pantalla.

Los siguientes procedimientos realizan la misma tarea e ilustran la diferencia existente entre la geometría de tortuga y la geometría de coordenadas:

```
TO CUADRADO1
  REPEAT 4 [FD 50 RT 90]
END

TO CUADRADO2
  SETXY 0 50
  SETXY 50 50
  SETXY 50 0
  SETXY 0 0
END
```

Entrar CUADRADO1 o CUADRADO2 después de retornar a DRAW dará exactamente el mismo resultado. Pero ¿qué sucede si se desea rotar los dos cuadrados en 30°? RT 30 CUADRADO1 funciona correctamente en el primer caso, pero el segundo procedimiento ha de ser reescrito por entero (porque especifica coordenadas de pantalla absolutas), y ésta no es una tarea banal. Pero existen ocasiones en las cuales el empleo de coordenadas es útil: como observará a partir de estos ejemplos, SETXY es mucho más rápida para dibujar líneas que FORWARD.

Otra característica de la geometría de tortuga es su "cortedad de vista". La tortuga se preocupa de un solo movimiento por vez, va construyendo las formas dando una serie de "pasos" cortos. Juguemos a la tortuga y consideremos cómo se construye un círculo. Imagine que usted es la tortuga: ¿qué necesitaría para producir una forma circular? Se movería hacia adelante una corta distancia y giraría un poco, y repetiría esta secuencia muchas veces. En términos de LOGO esto se define así:

```
TO CIRCULO
REPEAT 360 [FD 1 RT 1]
END
```

Este procedimiento funciona, pero se ejecuta muy lentamente. Se puede lograr mayor rapidez si no se dibuja la tortuga en cada paso. La instrucción del LOGO HIDETURTLE (ocultar tortuga) se utiliza para hacer invisible la tortuga: SHOWTURTLE (mostrar tortuga) la dibuja otra vez. Nuestro ejemplo en realidad dibuja un polígono de 360 lados; las líneas que lo forman son tan cortas que dan la impresión de una curva uniforme. De hecho, 360 lados sirven para dar la ilusión de un círculo; un polígono de 36 ya bastaría. De modo que el siguiente procedimiento dibuja un círculo aceptable y lo hace a una velocidad mayor:

```
TO CIRCULO
  REPEAT 36 [FD 10 RT 10]
END
```

Ahora intente escribir procedimientos que produzcan un semicírculo, un cuarto de círculo y una sexta parte de círculo, y combine dos de estas formas para conseguir una forma de pétalo.

Volviendo a jugar a la tortuga otra vez, ¿cómo se movería usted si deseara una trayectoria en espiral hacia un punto, en vez de un recorrido en círculo alrededor del mismo? Todavía se movería una corta distancia antes de girar, pero en este caso giraría más y más cada vez (o recorrería una distancia más corta para cada giro). Este procedimiento de espiral es un poco largo, ya que lo hemos elaborado de modo que emplee sólo las instrucciones que hemos introducido hasta el momento, pero servirá para demostrar ese principio:

```
TO ESPIRAL
  FD 10 RT 10 FD 10 RT 20 FD 10 RT 30
  FD 10 RT 40 FD 10 RT 50 FD 10 RT 60
  FD 10 RT 70 FD 10 RT 80 FD 10 RT 90
END
```

Alan Coode

Probando el LOGO
El juego *Demolition turtle* (La tortuga demoledora), de Anthony Ginn, introduce la tortuga para el suelo y ayuda al niño a dominar las relaciones espaciales. Un jugador coloca una torre de juguete en algún lugar del tablero, y elige la posición y el encabezamiento de partida de la tortuga. El otro jugador programa la tortuga para que choque contra la torre, usando la menor cantidad posible de instrucciones

Complementos al LOGO

Los editores de LOGO son muy similares, pero cada uno posee sus peculiaridades debido al teclado de cada máquina concreta. Consulte el manual de LOGO para la suya.
En todas las versiones LCSI la instrucción para editar el procedimiento CUADRADO es EDIT "CUADRADO (comillas antes del nombre del procedimiento, pero no al final).
Para situar la tortuga en (20,30) utilice SETPOS [2 30] en todas las versiones LCSI.

Respuestas ejercicios

Triángulo REPEAT 3 [FD 50 RT 120]
Pentágono REPEAT 5 [FD 50 RT 72]
Hexágono REPEAT 6 [FD 60 RT 60]
Rectáng. REPEAT 2 [FD 25 RT 90 FD 50 RT 90]
Paralelog. REPEAT 2 [FD 25 RT 70 FD 50 RT 110]
Rombo REPEAT 2 [FD 50 RT 70 FD 50 RT 110]

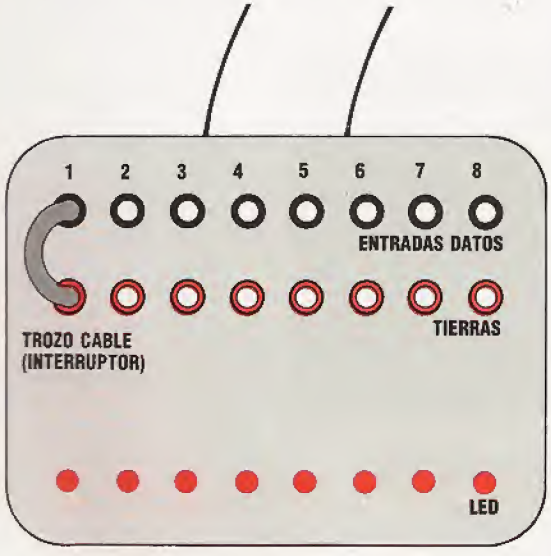
Estrellas:

1. REPEAT 5 [FD 50 RT 144]
2. REPEAT 8 [FD 50 RT 135]
3. RT 30 REPEAT 3 [FD 50 RT 120]
   PU LT 30 FD 29 RT 90 PD
   REPEAT 3 [FD 50 RT 120]
4. REPEAT 10 [FD 50 RT 108]

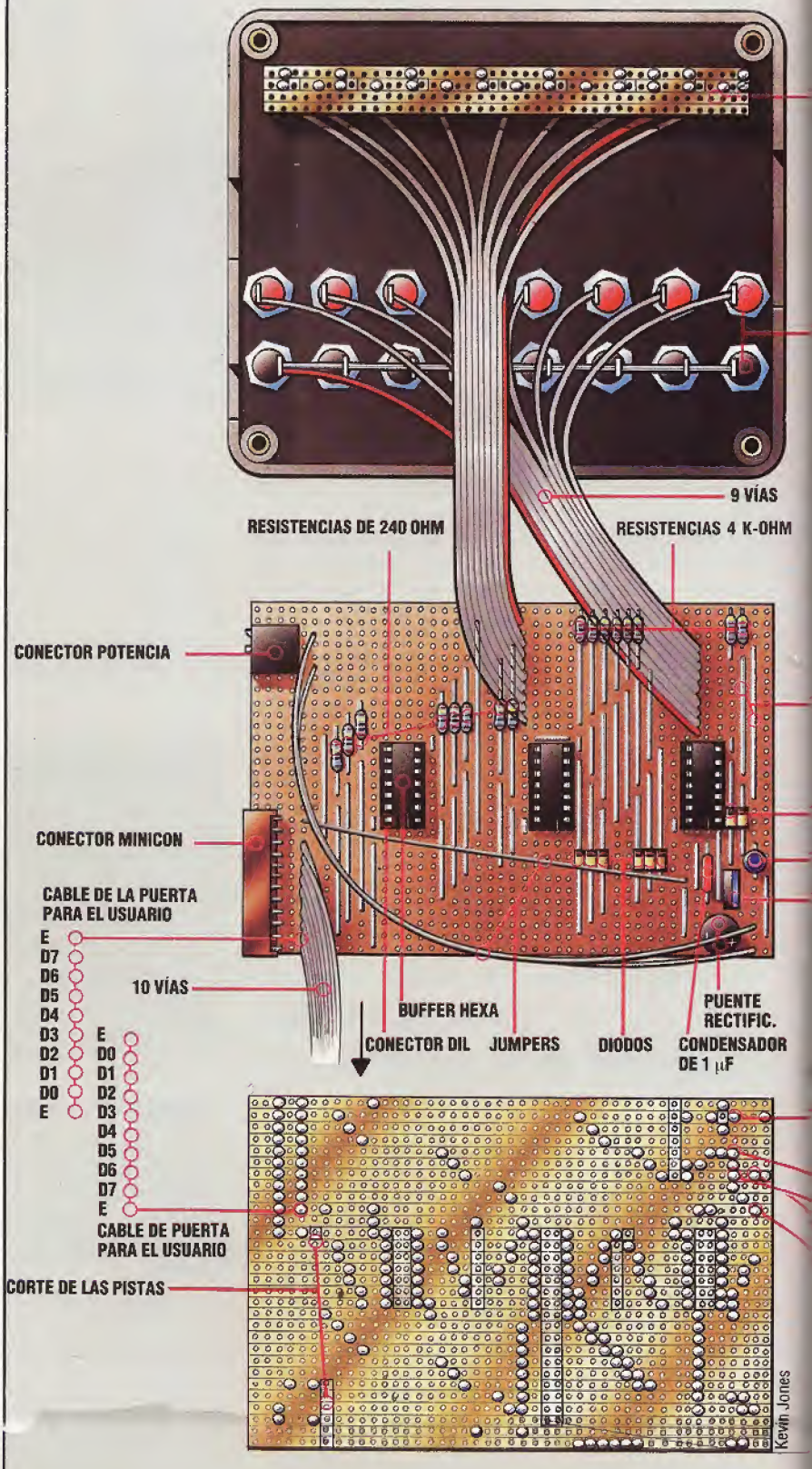
Todo en su lugar

Hemos llegado al momento de probar cómo funciona la caja diseñada en los capítulos anteriores de este apartado

Una vez construida la caja buffer con sus LED, podemos escribir algún software sencillo para comprobar su funcionamiento como cronometrador de reacciones. Aquí utilizaremos el bit 7 del registro de datos para el interruptor de entrada y los bits del 0 al 6 para los LED. El objetivo consiste en medir el intervalo transcurrido entre el instante en que el ordenador enciende los LED y el momento en que se pulsa un interruptor. Primero debemos conectar un interruptor entre la terminal de entrada 7 de la caja y la correspondiente toma a tierra; este interruptor puede ser de cualquier tipo puesto que sólo se utilizará para abrir y cerrar el circuito entre el bit 7 del registro de datos y tierra. Como interruptor se puede emplear, por ejemplo, un trozo de cable flexible. Conecte el interruptor, asegurándose de que esté interrumpiendo el circuito entre el bit 7 y tierra (el valor del bit 7 debe ser uno). El siguiente diagrama ilustra las conexiones:



El programa de prueba primero inicializa el registro de dirección de datos de manera que el bit 7 se utilice para entrada y todos los otros para salida, y después coloca a cero el registro de datos. Al cabo de una demora aleatoria, se encienden siete de los LED y se pone en marcha el contador de tiempo interno del ordenador. El cronometraje continúa hasta que el bit 7 se envía abajo, cuando se realiza la conexión. Los LED están cableados de modo tal que se iluminan cuando hay un cero presente en el registro de datos y se apagan cuando hay un uno.



Dentro de la caja

Pase un trozo de alambre estañado como muestra la ilustración a través de los contactos de los conectores. Suéldelo a cada uno de los contactos y compruebe la continuidad. Coja 20 cm de cable plano y separe tres cables, dejando un cable plano de nueve vías con la franja del borde de color. Pele y estañe los extremos de los cables. Suelde el cable de color en el conector cableado situado más a la izquierda. Ahora suelde los otros ocho cables restantes, por orden, en los contactos de los otros conectores. Pruebe la continuidad entre cada uno de los conectores y el final del cable al que estén conectados

LED

CONECTORES

Construcción de la placa

La placa ilustrada tiene 30 pistas con 45 agujeros y cabe exactamente en nuestra caja. Siga cuidadosamente las ilustraciones y no tendrá ningún problema para construir la placa. Utilice la menor cantidad posible de soldadura y tenga cuidado de no hacer puente entre las pistas; compruebe continuamente que esté colocando los componentes correctos en los lugares adecuados.

Los diodos, el condensador electrolítico, el puente rectificador y el regulador de voltaje se deben colocar todos en la dirección indicada; cualquier otra orientación los dañaría, de modo que estudie las marcas relativas a más y menos. Todos los componentes son sensibles al calor, por lo tanto no los "sobrecaliente" con el estaño.

Cuando instale el conector minicon y el de potencia procure colocar las patillas en los agujeros correctos de la placa, pero con cuidado de no torcer las patillas. Para los "cables de conexiones volantes" *(jumpers)* emplee los cables que quitó del cable plano.

Cuando todo esté colocado en su sitio en la placa, corte las pistas de cobre exactamente como muestra la ilustración. Para ello puede comprar una herramienta especial, o puede sostener una broca entre los dedos y hacerla girar en un agujero, cortando el cobre gradualmente. Suelde los cables planos a la placa. La orientación del conector y de los cables de los LED se indica mediante la franja de color, pero el cable de la puerta para el usuario exige un poco de atención; las dos líneas a tierra deben estar en los agujeros 1 y 10 (contando desde el borde de la placa) y las líneas de señal deben ir por orden en los agujeros del 2 al 9 de modo que la línea menos significativa esté más cerca del borde de la placa.

Por último, usando la placa como plantilla, corte ranuras en los lados de la caja para acomodar los conectores y el cable de la puerta para el usuario

ENLACE DE CABLES

DIODOS

CONDENS. ELECTROL. DE 1 µF

REGULADOR DE VOLTAJE

PUENTE RECTIFICADOR

REGULADOR DE VOLTAJE

CONDENSADOR ELECTROLÍTICO de 1 µF

```
10 REM * TIEMPO DE REACCION PARA EL BBC **
15 :
20 DDR=&FE62: REGDAT=&FE60
30 DDR=127: REM LINEAS 0-6 SALIDA
40 ?REGDAT=127: REM LED APAGADOS
50 :
60 CLS:PRINT "PREPARESE"
70 DEMORA=3000+RND(9000)
80 FOR I=1 TO DEMORA:NEXT:REM BUCLE DE DEMORA
85 FOR D=1 TO 200: NEXT D
90 :
97 REPEAT UNTIL ?REGDAT AND 128=1
100 ?FE60=0: REM ENCENDER LEDS
110 TIME=0: REM INICIALIZAR CONTADOR DE TIEMPO
120 REPEAT
130 UNTIL ?REGDAT AND 128=0: REM S/W ON
140 :
150 PRINT "TIEMPO INVERTIDO=";TIME/100;"SEG"
160 END
```

El programa emplea TIME, una variable reservada que devuelve un valor correspondiente a la cantidad de centésimas de segundo desde que TIME fuera puesta a cero por última vez. Haciendo uso del AND lógico en la línea 130 se aísla el bit 7 (valor 128) de modo que se pueda comprobar independientemente de los otros bits del registro de datos (véase p. 546). Cuando se conmuta el interruptor, el valor de un bit 7 pasa de uno a cero.

Se puede escribir un programa similar para el Commodore 64 utilizando su cronometrador interno, TI. Éste opera de manera distinta que TIME, devolviendo un valor en sesentavos de segundo desde que la máquina fuera encendida. Para emplearla debemos tomar el valor de TI al comienzo del intervalo a cronometrar y restarlo del valor de TI al acabar.

```
10 REM CMB 64 ** CRONOMETRADOR DEL TIEMPO DE REACCION **
20 :
30 DDR=56579: REGDAT=56577
40 POKE DDR,127: REM LINEAS 0-6 SALIDA
50 POKE REGDAT,127: REM LED APAGADOS
60 :
70 PRINT CHR$(147): REM LIMPIAR PANTALLA
80 PRINT "PREPARESE"
90 DE=3000+INT(9000*RND(1))
100 FOR N=1 TO DE:NEXT: REM BUCLE DE DEMORA
110 :
120 POKE REGDAT,0: REM ENCENDER LED
130 T=TI: REM TOMAR TIEMPO COMIENZO
140 IF PEEK(REGDAT) AND 128 <> 0 THEN 140
150 :
160 TM=(TI-T)/60: REM CALC INTERVALO
170 PRINT "TIEMPO INVERTIDO=";TM;"SEG"
180 END
```

En los próximos capítulos del curso analizaremos la construcción de salidas de corriente más alta, que son suficientes para activar motores eléctricos, y diseñaremos software para controlar motores bidireccionales y de velocidad variable.

Para probar el invento

1) Escriba un programa que encienda un LED cada vez en secuencia de izquierda a derecha.

2) Escriba un programa para hacer que los LED se iluminen en secuencia en las líneas del 0 al 6 (como en la pregunta 1), pero incluya un interruptor en la línea 7 para modificar la dirección de la secuencia. ¿Puede modificar su programa para que opere con un "tren" de tres LED?

3) Escriba un programa para simular el lanzamiento de un dado, utilizando seis LED y un interruptor.

4) Escriba un programa para simular la acción de un semáforo, utilizando tres LED.

5) Escriba un programa para contar la cantidad de "coches" (impulsos en un interruptor) que llegan mientras un semáforo está en rojo y para cambiar las luces cuando exceden de 10 o si ha transcurrido más de un minuto desde el último cambio.

Respuestas caja buffer

Éstas son las posibles soluciones a los Ejercicios de la página 1027:

Secuenciación de LED (I)

```
 10 REM CMB 64 VERSION 2.1
 20 DDR=56579: REGDAT=56577
 30 POKE DDR,225: REM TODAS SALIDA
 40 POKE REGDAT,225: REM TODOS LED APAGADOS
 50 GET A$
 60 FOR N=1 TO 7
 70 POKE REGDAT,255-(2^N)
 80 NEXT N
 90 IF A$="" THEN 50
100 END
```

```
 10 REM BBC VERSION 2.1
 20 DDR=&FE62:REGDAT=&FE60
 30 ?DDR=255: REM TODAS SALIDA
 40 ?REGDAT=255: REM TODOS LED APAGADOS
 50 REPEAT
 60 A$=INKEY$(1)
 70 FOR N=0 TO 7
 80 ?REGDAT=255-(2^N)
 85 FOR D=1 TO 200: NEXT D
 90 NEXT N
100 UNTIL A$ <> ""
110 END
```

Secuenciación de LED (II)

```
 10 REM CBM 64 VERSION 2.2
 20 DDR=56579: REGDAT=56577
 30 POKE DDR,127: REM L7 ENTRADA
 40 POKE REGDAT,255: REM TODOS LED APAGADOS
 50 GET A$
 60 FOR N=0 TO 7 STEP S
 70 POKE REGDAT,255-(2^N)
 80 NEXT N
 90 IF PEEK(REGDAT) AND 128=0 THEN S=-1
100 IF PEEK(REGDAT) AND 128=1 THEN S=1
110 IF A$="" THEN 50
120 END
```

```
 10 REM BBC VERSION 2.2
 20 DDR=&FE62:REGDAT=&FE60:S=1
 30 ?DDR=127: REM L7 ENTRADA
 40 ?REGDAT=255: REM TODOS LED APAGADOS
 50 REPEAT
 60 A$=INKEY$(1)
 70 FOR N=0 TO 7 STEP S
 80 ?REGDAT=255-(2^N)
 85 FOR D=1 TO 200: NEXT D
 90 IF REGDAT AND 128=0 THEN S=-1 ELSE S=1
100 NEXT N
110 UNTIL A$ <> ""
120 END
```

Secuenciación de LED (III)

Para el tren de tres LED, introduzca las siguientes modificaciones en la respuesta (2):

VERSION CBM

```
60 FOR N=2 TO 7 STEP S
70 POKE REGDAT,255-((2^N)+2^(N-1)+2^(N-2))
```

VERSION BBC

```
70 FOR N=2 TO 7 STEP S
80 ?REGDAT=255-((2^N)+2^(N-1)+2^(N-2))
```

CABLE DE PUENTE

Utilice 8 cm de cable aislado (una única línea del cable plano, por ejemplo) para conectar un enchufe rojo a un enchufe negro. Esto se denomina cable de parche y se utiliza para "puentear" un conector con otro. Haga ocho cables como éste

Kevin Jones

Lanzamiento de dados

```
 10 REM CBM 64 VERSION 2.3
 20 DDR=56579: REGDAT=56577
 30 POKE DDR,127: REM L7 ENTRADA
 40 POKE REGDAT,255: REM TODOS LED APAGADOS
 50 GET A$
 60 IF PEEK(REGDAT) AND 128 <> 0 THEN 60
 70 NR=INT(RND(1)*6)+1: REM SELECCIONAR NUM.
 80 POKE REGDAT,255-((2^NR)-1)
 90 IF PEEK(REGDAT) AND 128 <> 1 THEN 90
100 IF A$="" THEN 50
110 END
```

```
 10 REM BBC VERSION 2.3
 20 DDR=&FE62:REGDAT=&FE60:S=1
 30 ?DDR=127: REM L7 ENTRADA
 40 ?REGDAT=255: REM TODOS LED APAGADOS
 50 REPEAT
 60 A$=INKEY$(1)
 70 REPEAT UNTIL (?REGDAT AND 128)=0
 80 NUMERO=RND(6): REM SELECCIONAR NUMERO
 85 FOR D=1 TO 200: NEXT D
 90 ?REGDAT=255-((2^NUMERO)-1)
100 REPEAT UNTIL ?REGDAT AND 128=1
110 UNTIL A$ <>""
120 END
```

Semáforo (I)

```
 10 REM CBM 64 VERSION 2.4
 20 DDR=56579: REGDAT=56577
 30 POKE DDR,255: REM TODAS SALIDA
 40 POKE REGDAT,255: REM TODOS LED APAGADOS
 45 REM BIT2=ROJO,BIT1=AMBAR,BIT0=VERDE
 50 RJ=255-4: AM=255-2:VD=255-1
 60 GET A$
 70 POKE REGDAT,RJ
 80 FOR N=1 TO 200:NEXT:REM BUCLE DEMORA
 90 POKE REGDAT,RJ+AM
100 FOR N=1 TO 40:NEXT:REM BUCLE DEMORA
110 POKE REGDAT,VD
120 FOR N=1 TO 200:NEXT: REM BUCLE DEMORA
130 IF A$="" THEN 60
140 END
```

```
 10 REM BBC VERSION 2.4
 20 DDR=&FE62:REGDAT=&FE60:S=1
 30 ?DDR=255:REM TODAS SALIDA
 40 ?REGDAT=255: REM TODOS LED APAGADOS
 50 REM BIT2=ROJO, BIT1=AMBAR, BIT0=VERDE
 60 ROJO=255-4:AMBAR=255-2:VERDE=255-1
 70 REPEAT
 80 A$=INKEY$(100)
 85 FOR D=1 TO 200: NEXT D
 90 ?REGDAT=ROJO
100 FOR N=1 TO 200:NEXT:REM BUCLE DEMORA
110 ?REGDAT=ROJO+AMBAR
120 FOR N=1 TO 40:NEXT:REM BUCLE DEMORA
130 ?REGDAT=VERDE
140 FOR N=1 TO 200:NEXT:REM BUCLE DEMORA
150 UNTIL A$ <> ""
160 END
```

Semáforo (II)

Introduzca las siguientes modificaciones en la respuesta (4):

VERSION CBM

```
30 POKE DDR,127: REM L7 ENTRADA
65 T=TI: REM INICIALIZAR CRONOM.
75 IF PEEK(REGDAT) AND 128=0 THEN C=C+1
76 IF PEEK(REGDAT) AND 128 <> 1 THEN 76
77 IF C <=10 AND T-TI < 3600 THEN 75
```

VERSION BBC

```
30 ?DDR=127: REM L7 ENTRADA
75 T=0:C=0
95 REPEAT
96 IF ?REGDAT AND 128=0 THEN C=C+1
97 REPEAT UNTIL ?REGDAT AND 128=1
98 UNTIL T > 6000 OR C > 10
```

El "switch" o indicador

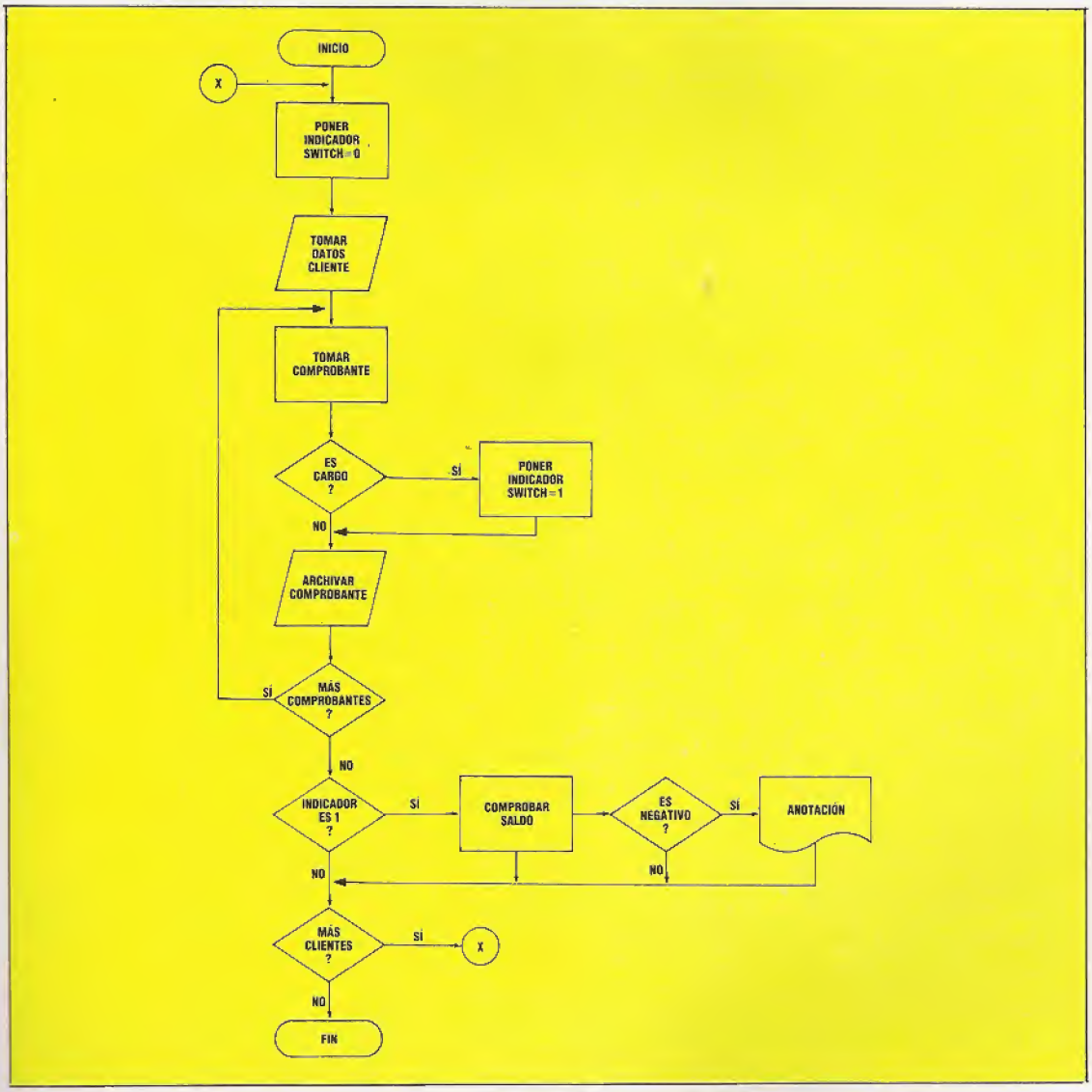
En esta ocasión nos referiremos a una variable de utilización esencial en el desarrollo de un programa: el "switch"

Un *switch* o indicador es una variable que puede asumir varios valores determinados, generalmente dos. Se utiliza en determinados puntos de un programa para indicar si el flujo del mismo ha pasado por un lugar concreto de la lógica o no; posteriormente, el mismo programa consulta el *switch* para saber por qué rama se ha llegado hasta el lugar donde se realiza la consulta.

En el ejemplo que presentamos se debe verificar el estado de cuentas de los clientes de un banco. Para ello hay que comprobar uno a uno los resguardos de operaciones de cada cliente. Cada comprobante representa una determinada operación de cargo o de abono. Una vez examinado cada resguardo, se archiva y se toma otro. Ahora bien, si entre los diferentes movimientos hay aunque sólo sea una operación de cargo, al acabar el proceso de todos los resguardos de ese cliente deberemos ver su estado actual, y en su caso señalar un posible saldo negativo antes de continuar con otro cliente.

En nuestro ejemplo, el *switch* actuará como una bandera indicadora de que el cliente tiene al menos una operación de cargo. Previamente al proceso de cada cliente, el *switch* se pondrá a valor 0. Este valor sólo se modificará si existe una operación de cargo, en cuyo caso tomará el valor 1. En la lógica del programa, una vez finalizado el proceso de todas las operaciones de un cliente, se consultará el estado de este indicador: si está todavía en su valor inicial significa que el cliente no tiene ninguna operación de cargo; si su valor es 1 significa lo contrario.

Obsérvese que, en general, un indicador sólo tiene dos estados, tal y como si se tratara de, por ejemplo, los dos estados de una bombilla, encendida o apagada.



En la cima del éxito

He aquí la historia de dos programadores adolescentes que se han hecho ricos a partir de la programación de una idea original

En el verano de 1984 Gremlin Graphics lanzó en Gran Bretaña un nuevo juego para ordenador llamado *Wanted: Monty Mole* (Se busca a Monty Mole). Se espera que el juego se convierta en un *bestseller*. De ser así, Peter Harrap, su programador, se encontrará con unas ganancias de varios miles de libras. Harrap tiene 19 años y *Wanted: Monty Mole* fue su primer juego comercial.

Peter Harrap pertenece a un pequeño grupo de "jóvenes genios" programadores, por lo general adolescentes, que pueden obtener considerables sumas de dinero si producen un juego que se coloque entre los diez mejores. Un muchachito de 14 años obtuvo casi 4 000 libras escribiendo software de juegos para el BBC Micro. Otro programador, de 17 años, fue obsequiado con un automóvil deportivo Lotus Esprit por la empresa de software para la cual trabajaba.

Al igual que la mayoría de los programadores de juegos, Harrap es autodidacta. El primer ordenador que tuvo fue un Sinclair ZX81 y, después de ver la reducida cantidad de software disponible que había en aquel entonces, decidió probar a escribir sus propios programas de juegos. Para hacerlo hubo de aprender el lenguaje máquina del Z80, puesto que el BASIC es demasiado lento para juegos recreativos. Harrap pronto dedicó sus atenciones al Sinclair Spectrum y se concentró en tratar de mejorar un juego que le resultaba particularmente interesante, el *Ant attack* de Quicksilva (véase p. 486). Harrap se metió en el código máquina y modificó el "paisaje" concebido en el programa. Quicksilva rechazó esta versión suya modificada, pero un amante local de los ordenadores, Ian Stewart, que estaba buscando un nuevo talento para su pequeña empresa de software, quedó impresionado por la cinta que Harrap le hizo llegar. El juego de Harrap violaba el copyright del *Ant attack* original, por lo que la empresa de Stewart, Gremlin Graphics, no lo aceptó. Sin embargo, la empresa necesitaba un programador muy cualificado para el Spectrum, de modo que contrataron a Harrap de inmediato.

Todas las empresas de software buscan juegos de temática interesante e inusual con los que esperan trepar hasta el primer puesto de las listas de software. Gremlin Graphics concibió la idea de *Wanted: Monty Mole* con la esperanza de que su tópico tema de la huelga de los mineros atrapara la imaginación del público. La idea la puso en práctica Harrap, quien escribió un programa para ejecutar el juego en el Sinclair Spectrum.

La programación del juego le llevó a Harrap cuatro meses. Empezó escribiendo una rutina generadora de sprites para poder diseñar rápidamente las diversas formas de los gráficos, luego dibujó la mina de carbón y por último programó las reglas que determinan los movimientos de los personajes.

Ian McKinnell

El detalle de los detallistas
La presentación del punto de venta tal vez sea la más importante de todas las interfaces para el usuario; aquí se combinan los efectos de la publicidad, los reportajes, el empaquetamiento, la comercialización y el atractivo del producto para influir en la decisión del cliente. La disponibilidad es, obviamente, un factor crucial para las ventas, de modo que conseguir que el producto se distribuya a través de cadenas de tiendas al detall es uno de los objetivos básicos de todos los editores de software

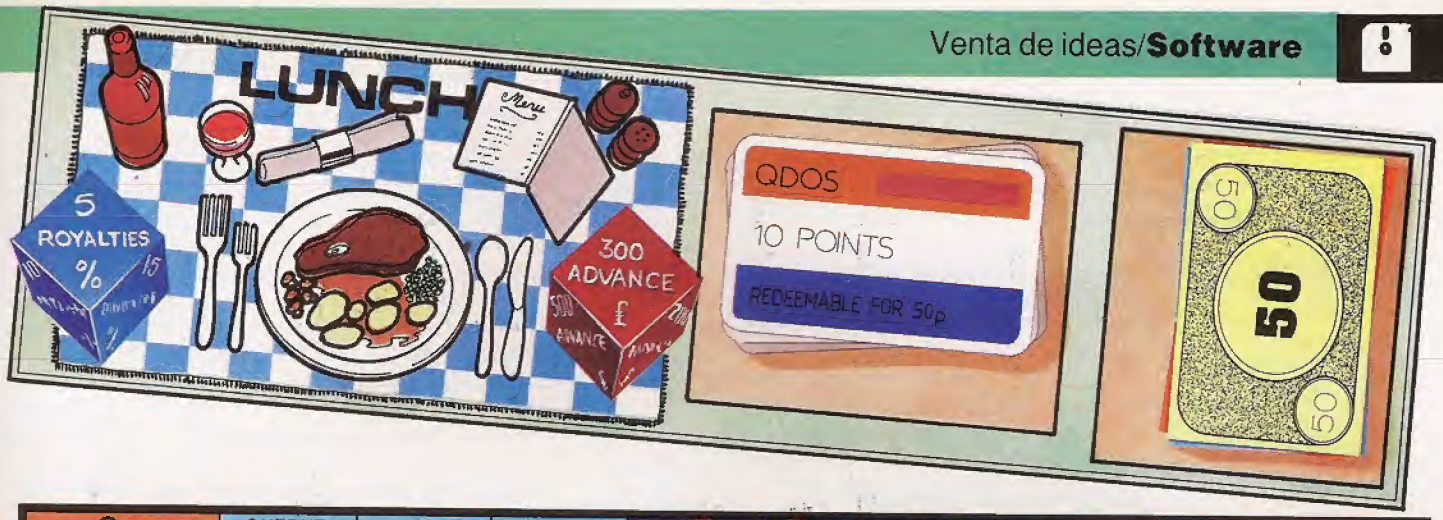
El juego fue objeto de reportajes en las televisiones de toda Gran Bretaña debido a la actualidad de su temática, y se convirtió en un éxito de ventas. Los informes de los medios de comunicación no ignoraron la ironía de que el padre de Harrap, que trabajaba en la carbonera local, se hallaba él mismo en huelga, ni tampoco el hecho de que Gremlin Graphics se había comprometido a donar cinco peniques por cada cassette vendida al Miners' Welfare Fund (Fondo para el bienestar de los mineros).

Gremlin Graphics le paga a Harrap un *royalty* o porcentaje sobre cada cassette vendida. El montante de estos royalties varían de una empresa a otra. La mayoría de las empresas abonan royalties según una escala variable: el 5 % para los primeros 3 000 ejemplares vendidos, por ejemplo, y el 7 % para los 10 000 siguientes. El porcentaje se calcula sobre el precio neto, que es el que percibe la empresa de software por parte del distribuidor, o bien sobre el precio bruto, que es el que paga el público.

La mayoría de las empresas también les abonan a los programadores un *anticipo*. El mismo se abona a cuenta de los royalties, es decir, que será deducido de los primeros royalties que se abonen. Sin embargo, muchas empresas no les pagan royalties a sus programadores, sino que optan por un único pago al principio. Los programadores han de sopesar detenidamente esta clase de arreglos, porque pueden resultar una pérdida de beneficios en el caso de que las ventas sean elevadas.

En un acuerdo basado en royalties, el programador conserva los derechos de copyright del juego. Esto significa que en caso de cualquier litigio es el

Consiguiéndolo
Se cuentan por miles los usuarios de micros personales que están intentando escribir paquetes de software; algunos de ellos llegan realmente a alcanzar el éxito, encontrando un editor y ganándose algún dinero. Puede que nuestro juego imaginario exagere un poco el elemento de cambio en el proceso, pero con toda seguridad, es una lotería...



PILLAJE

El juego de Juegos

- Se estimula a los jugadores para que engañen y estafen a los otros y para que infrinjan la mayor cantidad de reglas posible.
- El objetivo del juego consiste en echar a los demás jugadores del mismo mediante la acumulación de todos los QDOS y el dinero. Esto se consigue llevándole los detalles de su IDEA a un EDITOR (PUBLISHER) y llegando a un acuerdo durante el LUNCH. Su primer acuerdo solo le reporta un royalty; los acuerdos subsiguientes le reportan un royalty y un anticipo inmediato.
- Una vez hecho el acuerdo, su cassette sale a la venta en un BUGMART, obteniendo beneficios abonados por todos los jugadores que aterricen allí.
- Cada vez que pulse RETURN, usted obtiene una nueva IDEA y recoge sus royalties, que se calculan multiplicando sus QDOS por su porcentaje de royalty.
- Cuando caiga en el teclado (GOTO KEYBOARD), debe ir directamente; no debe pulsar RETURN, ni recoger sus ganancias, y debe quedarse allí hasta que le salga un seis o consiga robar la IDEA de alguien.
- BERGSTROM toma su último juego y lo empaqueta con su nuevo micro. Ello hace que sus ingresos por ese juego se dupliquen.

Steve Cross

programador el demandante, o bien el demandado. Si el programador recibe por su juego una única cantidad global, la empresa de software por lo general también comprará el copyright.

Además de los royalties, Harrap también percibe una cantidad fija, pero tiene un contrato exclusivo con Gremlin Graphics por un año y no puede trabajar para ninguna otra firma de software. Muchos contratos también incluyen una cláusula de opción en virtud de la cual la empresa de software tiene el derecho exclusivo de preferencia para aceptar o rechazar cualquier idea o programa.

Para que un juego tenga éxito, una empresa persigue unas ventas de al menos 15 000 ejemplares durante su vida, que suele ser de cuatro o cinco meses. En el caso de *Wanted: Monty Mole*, la empresa inicialmente reprodujo 10 000 ejemplares para cada máquina (Spectrum y Commodore 64), con la opción de reproducir más. Cuando se lanzó el juego, Ian Stewart predijo unas ventas de 20 000 para cada máquina y afirmó: "Creo que definitivamente se convertirá en el número uno".

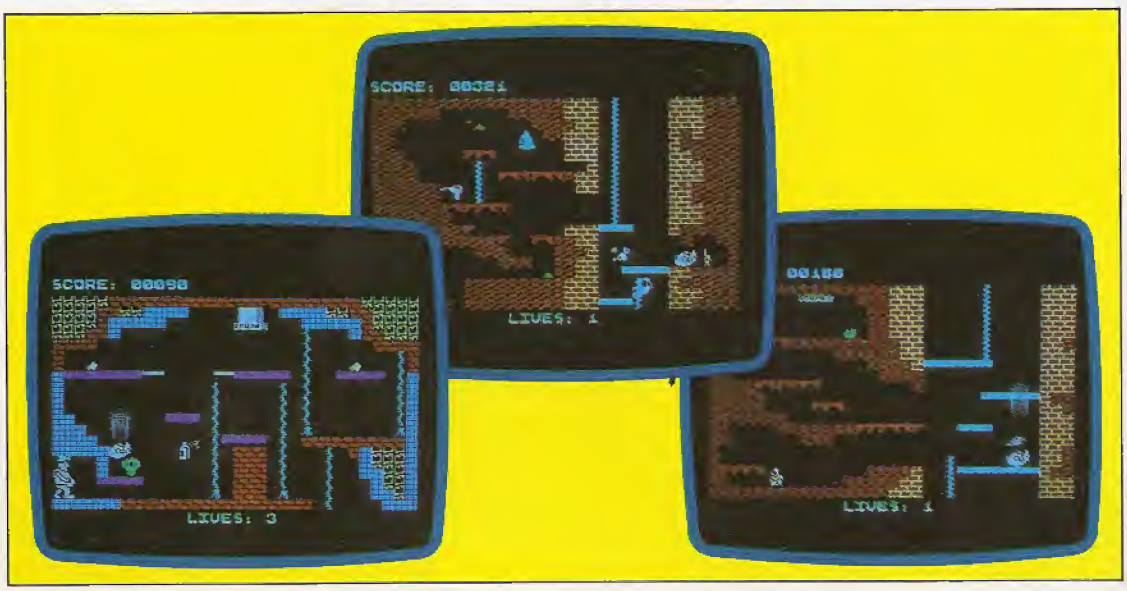
Christopher Kerry es otro programador a quien el éxito le ha sonreído incluso antes. A los 17 años abandonó los estudios para trabajar por libre para la empresa de software House of Thor. Kerry escribió el juego *Jack and the Beanstalk* en el ordenador Spectrum que tenía en su casa de Sheffield. Sin publicidad, y después de haber estado a la venta durante apenas dos meses, se habían vendido "decenas de miles", según la empresa (que no estaba preparada para revelar la cifra exacta). En realidad House of Thor tenía una actitud muy protectora hacia Kerry, aduciendo que temían que alguna otra empresa estuviera tratando de "pescarlo". Además de Christopher Kerry, House of Thor utiliza a unos 15 programadores, la mayoría adolescentes.

Al igual que todas las empresas de software, House of Thor acepta de buen grado los programas que le envía la gente para su evaluación y, si parecen prometedores, la empresa se pone en contacto con el programador ese mismo día. Una respuesta rápida es de vital importancia, porque bien podría ser que el mismo software se le hubiera ofrecido también a otras empresas. House of Thor sólo se interesa por juegos de estilo recreativo y éstos sólo se pueden escribir realmente en lenguaje máquina. La empresa también busca temas originales y, por lo tanto, no toma en cuenta aquellos programas que sean prácticamente copias de juegos ya existentes: uno no puede limitarse a modificar los gráficos o cambiarles los nombres a los monstruos, por ejemplo. Aparte de las posibles acciones legales que ello podría suponer, es poco probable que esta clase de juegos se venda bien.

Salamander es una gran firma de software que tiene contratados alrededor de 40 programadores. Chris Holland, su director gerente, les hace sugerencias a los potenciales escritores de software: "Cuando uno puede hacer realmente el agosto es escribiendo software para las máquinas nuevas apenas éstas salen al mercado. Nosotros recibimos de buen grado software para los micros Amstrad, Atmos y los MSX. Con uno como el Spectrum ya es más difícil, a menos que se trate de una idea completamente nueva". La empresa posee una gama de alrededor de 50 títulos para la mayoría de los ordenadores personales populares, no tan sólo para el Commodore 64 y el Spectrum.

En vez de intentar venderles los programas a las firmas de software, hay algunos programadores que crean sus propias empresas. Llamasoft y Bug-Byte son apenas dos ejemplos de algunas casas de software que han sido fundadas por jóvenes programadores de juegos. Lamentablemente, se necesita un capital bastante grande para empezar. La empresa de Ian Stewart, Gremlin Graphics, empezó con un solo juego, *Potty Pigeon*, pero pronto esperan contar con una gama de cinco programas. Los cuatro directores de la empresa han puesto dinero de su propio bolsillo para pagar la publicidad y la distribución; un anuncio en color de una página en una revista de ordenadores vale más de mil libras si se incluyen los costos de producción. Ni siquiera una gama de juegos de éxito constituye una garantía de seguridad financiera. Imagine (véase p. 559) se declaró en quiebra en julio de 1984 debiendo más de 1 millón de libras, a pesar de que en un momento dado la empresa estuvo vendiendo software de juegos por valor de 300 000 libras al mes.

Montar una empresa de software puede ser arriesgado; además de buenos juegos, también se necesita asesoramiento financiero. Pero cualquiera puede enviarle un programa a una empresa de software para ver "qué pasa". Quién sabe, tal vez usted haya escrito el próximo número uno en ventas.



Liz Heaney

Los pozos
El programador que escribió *Wanted: Monty Mole* es el arquetipo del genio de software: joven, autodidacta, brillante y bien remunerado. El juego en sí mismo dista mucho de ser un caso típico; adopta una posición petulantemente antisindical respecto a la huelga de los mineros de 1984, pero cada ejemplar que se venda supondrá la donación por parte de los editores de cinco peniques para el Miners' Welfare Fund (Fondo para el bienestar de los mineros)

Interface sonora

En esta ocasión analizaremos parte del hardware y del software que permiten al usuario utilizar la interface MIDI

El Micon, o *MI*DI *con*troller (controlador de MIDI), lo produce XRI Systems y es bastante representativo de los paquetes MIDI que producen las casas de software más modestas. Está diseñado para utilizarse con el Sinclair Spectrum, y se compone de una interface MIDI y una cassette de software. Su facilidad de secuenciador proporciona ocho pistas separadas, cada una de ellas con capacidad para 2 951 eventos (notas, pausas, etc.). La música se entra en el teclado del sintetizador conectado en interface y se utiliza la tecla Space del Spectrum para definir la cantidad de eventos a ejecutar.

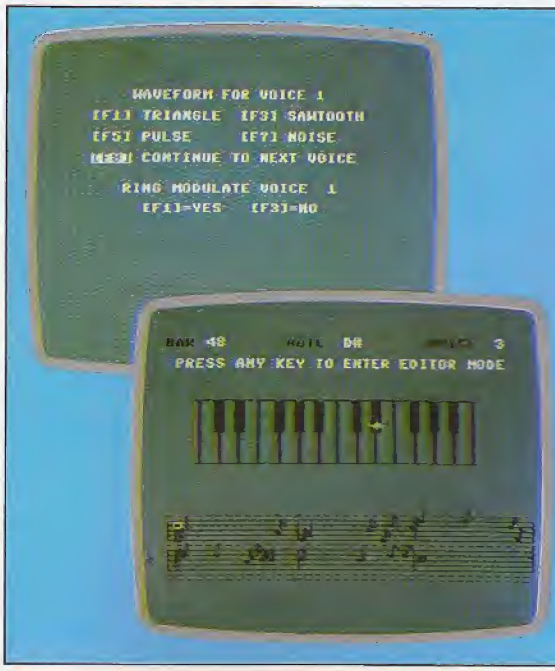
Mientras se va utilizando el sintetizador, la música se va desarrollando en la pantalla en una forma tosca de notación estándar con pentagrama de cinco líneas. Ésta proporciona la clave de *fa* o de *sol*, la gama completa de duraciones (incluyendo notas puntuadas), sostenidos y bemoles, e indicadores de staccato (notas muy cortas). La notación es pobre para indicar pausas (aquellas duraciones donde *no* se producen sonidos de notas juegan un papel esencial en cualquier composición musical) y los rabos de las notas apuntan siempre hacia arriba, no hacia arriba o hacia abajo según el lugar ocupado en el pentagrama. Además, las frases de notas cortas no se reorganizan de forma racional en grupos que reflejan el tiempo o ritmo implícito de la música. El resultado de todo ello es difícil de leer, excepto como guía aproximada, pero se puede imprimir mediante la impresora ZX.

La utilidad de la visualización se hace evidente cuando se trata de editar la composición. La música se divide en compases y, entrando un número de compás, se pueden eliminar, insertar o modificar notas individuales especificando la nueva nota. Se pueden borrar compases enteros y se pueden repetir agrupamientos de compases en cualquier lugar dentro de la secuencia. Es posible guardar en cinta hasta 10 secuencias (casi 24 000 notas), junto con las características apropiadas para su reproducción. La velocidad de la reproducción se puede definir dentro de cuatro milisegundos o controlar con una máquina de ritmos utilizando un conector SYNC IN en la interface.

El Micon también secuencia en tiempo real. En efecto, "escuchará" una ejecución en un sintetizador compatible con la MIDI y entrará todos los datos en la memoria del Spectrum. No visualizará una notación de la ejecución, pero es útil en otros dos sentidos. En primer lugar, permite que los músicos de teclado oigan una grabación de su propia actuación, sin tener que preocuparse por establecer niveles de entrada en cinta; su instantaneidad será especialmente valiosa para la educación musical. En segundo lugar, siempre que la música requerida esté "bajo los dedos" del ejecutante, alivia gran parte del tedio de la secuenciación de golpes de compás. No obstante, el ejecutante aún necesita de un metrónomo o patrón de máquina de ritmos junto al cual tocar, con el fin de llevar un tempo constante, porque de lo contrario la frase se reproducirá con todos los errores de ritmo del ejecutante. Al igual que con cualquier otro sistema MIDI, los resultados obtenidos con el Micon dependerán de la sofisticación del diseño del sintetizador con el que esté conectado.

El paquete MIDI de Jellinghaus Music System vale un poco menos que el Micon y además está diseñado para utilizarlo con una gama más amplia de máquinas: el Apple II, el Commodore 64 y el Sinclair Spectrum. Básicamente es similar al Micon en cuanto a que la información musical se entra desde el teclado del sintetizador conectado en interface. La principal diferencia es que las frases sólo se pueden entrar en tiempo real y que la visualización en pantalla de la música no responde a la notación estándar de pentagrama de cinco líneas.

La organización de la música en compases se realiza especificando los tiempos, es decir, cuántas negras o corcheas entran en cada compás. Hay cuatro tipos de compás disponibles: tres, cuatro o cinco negras por compás (3/4, 4/4 o 5/4 tiempo), o siete corcheas (7/8) y, al igual que con el Micon, el compás de la ejecución se ha de coordinar mediante un metrónomo separado o una máquina de ritmos. Si la actuación en tiempo real se desvía en velocidad, entonces los datos de entrada se convertirán en una cadena de notas "abiertas", sin compás. Después de que la frase esté lista para su reproducción, no habrá ningún problema hasta que no se produzca algún agrupamiento que no sea de 3/4, de 4/4, de



Control de sonido

Aprovechando el chip de sonido del Commodore 64, se puede escribir un programa en BASIC para entrar música con un formato visual mediante una palanca de mando y hacer que la misma se reproduzca. En el ejemplo que vemos aquí aparece una luz en un teclado gráfico y se puede llevar hacia arriba de la escala pulsando *forward* en la palanca de mando, o hacia abajo de la escala tirando hacia atrás. Cuando uno llega a la nota deseada, mueve la palanca de lado a lado para seleccionar la duración de la nota. Ésta se "graba" cuando se acciona el pulsador de disparo, y se está entonces preparado para introducir la siguiente nota de una frase

Liz Heaney

1 Control por ordenador
El Commodore 64 proporciona gestión de memoria y operación conducida por menú del sistema de música a través de software. Se puede utilizar la unidad de disco, o la grabadora de cassette, para efectuar grabaciones digitales de frases y almacenarlas para su uso ulterior

5/4 ni de 7/8. Apenas sucede esto, falla la facilidad de secuenciación. Esto también significa que no se puede entrar un tiempo como 12/8, aunque la velocidad sea perfecta, una omisión lamentable, puesto que 12/8 es un tiempo de *swing* comúnmente utilizado en jazz, rock, funk y reggae.

Siempre y cuando se haya dado entrada a una frase correctamente, los datos se visualizan en la pantalla en columnas, mostrando la octava en la cual se produce el evento de nota, la altura de la nota dentro de la octava, la duración en términos musicales (negras y corcheas), *gate-time* (un parámetro que se utiliza para otorgar un cierto fraseado a la línea musical) y los valores de "sensación táctil" del teclado, de 0 a 9.

Este tipo de visualización de hecho proporciona más información sobre una base de nota por nota que el empleo de la notación estándar que utiliza el Micon, que jamás ha incorporado ni *gate-time* ni niveles de velocidad, al menos no en forma numérica. Ello se debe a que la capacidad para especificar tales parámetros se ha introducido en el vocabulario musical sólo a raíz del advenimiento de la electrónica.

No obstante, muchos músicos sin ninguna clase de entrenamiento formal desarrollan enseguida una habilidad para mirar una sección de notación estándar y, mediante el empleo de la forma "subida y bajada" de las notas esbozada en el pentagrama, se hacen alguna idea sobre cómo sonará una melodía, o qué grupo de notas de un acorde se suman armónicamente. Esto significa que, en la práctica, la notación se puede leer, y una frase musical "escuchar" mentalmente en tiempo real. Las visualizaciones de columnas pueden ser excelentes para comprobar datos, pero son pocas las personas que pueden tararear una melodía a partir de ellas. Un sistema ideal incluiría los dos tipos de visualización; pero, al no ser éste el caso, la torpe versión del Micon de la notación estándar es más eficaz e idónea para desarrollar aptitudes musicales.

Sintetizadores conceptuales

Puede que el principal problema de cara al usuario de un ordenador personal con inclinaciones musicales no sea el de hallar el paquete MIDI correcto, sino el de obtener un sintetizador compatible con la MIDI que sea suficientemente sofisticado como para hacer el mejor uso de la interface sin que por ello el usuario necesite un segundo préstamo para pagarlo. Si exigimos que el sintetizador sea capaz de manipular el 50 % de las pistas de refuerzo de un disco *pop* sencillo típico de hoy, nos encontramos con que uno de tales instrumentos cuesta entre 200 000 y 300 000 pesetas. Por debajo de esta gama de precios, la mayoría de los sintetizadores son sumamente limitados en cuanto a su aplicación.

Una posibilidad a tener en cuenta es lo que se ha dado en llamar *sintetizador conceptual*: típicamente, un paquete de software avanzado y un teclado periférico que utiliza el potencial para generación de sonido de un microordenador, en vez de una interface a un sintetizador que es probable se haya diseñado antes de que se desarrollara la MIDI.

Un ejemplo es el PDSG (*Programmable Digital Sound Generator:* generador digital y programable de sonido). El mismo se conecta en interface con el microordenador BBC Modelo B. Incluye un teclado musical sensible a la velocidad, de 61 notas y dos pedales de pie. El teclado sensible representa la di-

2 Six-Trak, de Sequential Circuits
Potente sintetizador con una grabadora de cinta de pistas múltiples incorporada.
El Six-Trak (seis pistas) se llama así porque puede grabar en seis pistas.
A diferencia de muchos sintetizadores de esta clase, permite que cada voz musical tenga un sonido diferente, lo que posibilita la creación de sonidos complejos. Cada voz se puede controlar por frecuencia, timbre, forma de onda, ligadura de frecuencia y flexión de nota

3 Model 64, de Sequential Circuits
Enchufando el Model 64 a la puerta para ampliación de memoria del Commodore 64 es posible incorporar un sintetizador equipado con MIDI y la memoria, almacenamiento en disco o cassette y visualización en vídeo del ordenador a un sistema musical. El Model 64 almacena información de compás, altura y modulación para hasta 4 000 notas en modalidad de grabación. Para la reproducción, la interface puede enviar la señal digital exactamente tal como fue recibida desde el teclado (tiempo real), o corregirla para una indicación determinada de tiempos (golpes de compás)

Ian McKinnell

ferencia entre un tacto sensible y uno "muerto" en la ejecución, y los datos de sensibilidad de la ejecución en tiempo real se pueden almacenar para grabación y reproducción. Posee 32 unidades generadoras de sonido, en lugar de osciladores, cada una de las cuales puede tener hasta 11 características definidas. Si así se requiere, se pueden utilizar las 32 unidades para producir una única nota. Esta facilidad sola, en manos de un usuario capaz, le confiere al PDSG una riqueza y una variedad de sonido al nivel de la mayoría de los sintetizadores que hemos reseñado en el recuadro.

Si se le asigna una sola unidad generadora a cada nota, entonces una frase programada puede constar de 32 líneas individuales. Por otra parte, se puede utilizar cierta proporción de generadores para material secuenciado, y tocar los restantes en tiempo real junto a la frase. Las características de forma de onda se visualizan en la pantalla, ofreciendo la oportunidad de analizar los sonidos visualmente, un apoyo inestimable para adivinación auditiva, y un factor gracias al cual el PDSG resulta extraordinariamente apropiado para la educación musical.

La principal desventaja del PDSG es su representación del sonido en la fase de conversión de digital a analógico. El oído y el cerebro humanos pueden interpretar sonidos a través de una anchura de banda entre 20 Hz y 20 KHz. Los sonidos naturales, incluyendo aquellos producidos por instrumentos musicales acústicos, son activos dentro del total de esta anchura de banda y más. El PDSG, sin embargo, puede representar sonido sólo dentro de una anchura de banda de hasta 12 KHz. A consecuencia de ello, su calidad de sonido es comparable con la de un sistema de alta fidelidad doméstico adecuado, y los fabricantes dan por sentado que para completar el sistema se utilizarán altavoces y un amplificador *hi-fi* domésticos. La mayoría de los músicos sin sintetizador quedarían consternados si ésta fuera su única opción de amplificación. La firma británica Clef Products tiene planificadas versiones del generador digital y programable de sonido (PDSG) para conectar en interface con otros microordenadores.

La MIDI se ha considerado como una brecha porque les proporciona a los usuarios de microordenadores acceso a verdaderos sintetizadores musicales. El sistema PDSG es suficientemente avanzado en muchos aspectos como para que los propietarios de sintetizadores consideren, en cambio, la posibilidad de comprarse un microordenador y un "sintetizador conceptual".

Colaboración a la música

El desarrollo de la interface MIDI les proporciona a los usuarios de microordenadores una gama de posibilidades para hacer música. Pero, al mismo tiempo, existe el riesgo de adquirir un paquete caro (la interface propiamente dicha, el software adicional y un sintetizador) sólo para quedar abrumado por los intrincados detalles del sistema. Una alternativa es comenzar con un sistema de música económico para familiarizarse con los fundamentos de la música electrónica. Tal sistema, por supuesto, debe ser suficientemente bueno como para que resulte satisfactorio desde el punto de vista musical y despierte el interés por las demás posibilidades de la música electrónica. Un buen "paquete de iniciación" es el Ultisynth 64, basado en cassette y producido por Quicksilva. Éste explota el chip SID *(sound interface device)* del Commodore 64 y sus tres osciladores.
Utilizando el paquete, cada una de las teclas del teclado del Commodore se convierte en un control independiente para generar y definir sonido. Están disponibles las cuatro formas de onda básicas (sinusoidal, cuadrada, triangular y aserrada), junto con la posibilidad de establecer incrementos para definir las características de ataque-sostenimiento-decaimiento-liberación (ADSR o envoltura). Los sonidos se pueden filtrar (es decir, se puede restar de la salida una anchura de banda de frecuencias especificada) para caracterizar al sonido aún más. La modulación de anillo (un proceso que proporciona la suma y la diferencia de dos frecuencias cualesquiera) también se incluye. Esto es útil para producir sonidos tipo campana bastante auténticos. Además se puede escribir el ritmo, incorporar ritmos preestablecidos y secuenciar 2 048 notas.
Las facilidades del Ultisynth se asemejan mucho a las del VCS 3, un sintetizador "clásico", controlado por voltaje, de finales de los sesenta.
Otro sistema basado en cassette adecuado para el principiante es el Multisound de Romik, que es muy parecido al Ultisynth en cuanto a sus facilidades de control, pero que ofrece la visualización de un teclado musical. En este teclado las posiciones se seleccionan utilizando un cursor y las notas se definen con información entrada mediante el teclado alfanumérico del ordenador.

Opciones de elección

En el tema de guiar al usuario en un programa veremos aquí dos técnicas típicas: el sistema de menús y el sistema de comandos

Los menús pueden ser simples listas de ítems numerados o pueden ser pantallas llenas de elaborados iconos, pero el principio que rige su utilización es siempre el mismo. El menú se emplea cuando el programa alcanza, en su lógica, una bifurcación de caminos múltiples; entonces se solicita al usuario que elija qué ruta tomar a partir de una lista de las opciones disponibles que se visualiza en la pantalla. Los programas basados en menús tienden a asumir estructuras arborescentes: el usuario entra en el árbol por la "raíz" y es guiado mediante las opciones del menú hacia una de las "hojas", donde se halla la información o función.

La principal ventaja de este enfoque es que el usuario necesita poco o ningún conocimiento acerca de la estructura del programa, porque el camino está bien "señalizado" a través de toda la ruta. No obstante, a los usuarios más experimentados la labor de ir abriéndose paso a través de una cadena de menús les resulta tediosa cuando se trata de tareas que se repiten con frecuencia. Los novatos, asimismo, pueden tener dificultades con la estructura arbórea; corregir una decisión errónea implica ir trabajando hacia atrás a través de todos los menús, hasta el punto en el cual se incurrió en el error, volver a entrar la opción correcta y después seguir adelante a partir de allí. El Prestel es una estructura particularmente "profunda" de este tipo y los usuarios se encuentran frecuentemente con este problema. No es en absoluto necesario que los menús conformen un árbol; se pueden organizar en red mediante la utilización de bucles.

Diseñar un sistema de menú puede ser difícil, a pesar de que la programación propiamente dicha es relativamente fácil. El principal problema es que se debe especificar claramente todo el programa completo antes de escribir ninguna instrucción. (Ésta es, de todas maneras, una buena costumbre, pero no siempre es fácil.) Agregar nuevas funciones en una etapa ulterior puede implicar la modificación de varios menús anteriores del programa, y esto puede requerir una importante reestructuración. Cuando se diseña el programa se debe incluir toda la lógica de los menús en una única rutina que llame a las rutinas de las "hojas" cuando se llegue a las mismas. La rutina de menú se puede considerar, por consiguiente, como una forma más compleja de la rutina de control normal, con todas las bifurcaciones internas controladas por el usuario. Ello produce un diseño pulcro y sirve para separar la lógica de control de las partes funcionales del programa, permitiendo desarrollar y depurar cada una de ellas de forma independiente.

El flujo del programa no seguirá un patrón establecido. Para cada menú a lo largo de la ruta, la rutina de lógica de los menús pasa un conjunto de indicaciones para el usuario a una rutina que las coloca en sus lugares concretos dentro de una "ventana" de menú. Las "ventanas" probablemente contendrán un mensaje de encabezamiento de pantalla que visualice un título y cualquier otra información necesaria acerca del menú, un "pie" que explique cómo hacer la elección (con espacio suficiente para la respuesta del usuario) y las opciones del menú propiamente dichas. En general, el trazado de menú más eficaz posee hasta ocho opciones visualizadas en una columna, con el código de respuesta (número, letra, símbolo mnemónico, etc.) a la izquierda de cada ítem.

La rutina de menús llama a una rutina de entrada, tal vez pasándole las condiciones que se deben especificar para una entrada legal, y acepta a su vez la respuesta del usuario. Luego interpreta esta respuesta (por lo general, una única pulsación de tecla) y o bien le pasa el control al siguiente menú, o llama a la rutina de aplicación apropiada si se trata del último menú de la cadena. Una vez que se ha ejecutado la rutina, se puede visualizar el menú desde el cual fue llamada, o el control puede pasar a alguna otra parte del programa.

Los menús exigen muchísimo texto para encabezamientos, pies e indicaciones, pero gran parte del mismo se repite para cada "ventana" de menú. La explicación sobre cómo elegir una opción del menú (la instrucción de "ayuda": *help*), una opción que ofrezca una salida al menú raíz, y otras opciones posibles puede que se necesiten en varios menús diferentes. De ser éste el caso, se puede ahorrar espacio y hacer que la lógica sea más clara si todas las solicitudes se retienen en una matriz de seres (o en un archivo en disco de acceso directo), desde la cual podrían ser llamadas mediante su número de índice. Diseñe la rutina de visualización de menús de modo que acepte referencias a esta matriz y que visualice los encabezamientos, pies, solicitudes, etc., apropiados.

Un sistema activado por instrucciones es aquel que posee una gama de instrucciones que están a disposición del usuario en todas las etapas del programa. Cada instrucción va directamente a una subrutina que lleva a cabo la función requerida. Este sistema se debe diseñar para inspeccionar todos los input y discenir si se trata de datos o de una instrucción del programa. La diferencia por lo general la señala el usuario mediante la pulsación de una tecla determinada antes de cada entrada de instrucciones. La tecla control se usa frecuentemente para este fin. Un procesador de textos, por ejemplo, puede aceptar la palabra *save* (guardar, salvar) como una palabra más del texto, pero también es posible que la interprete como una instrucción de almacenamiento si antes de digitar la palabra se pulsa la tecla Control.

En un sistema activado por instrucciones, el "árbol" es muy bajo y ancho, y se utiliza una única rutina, que actúa a modo de programa de control,

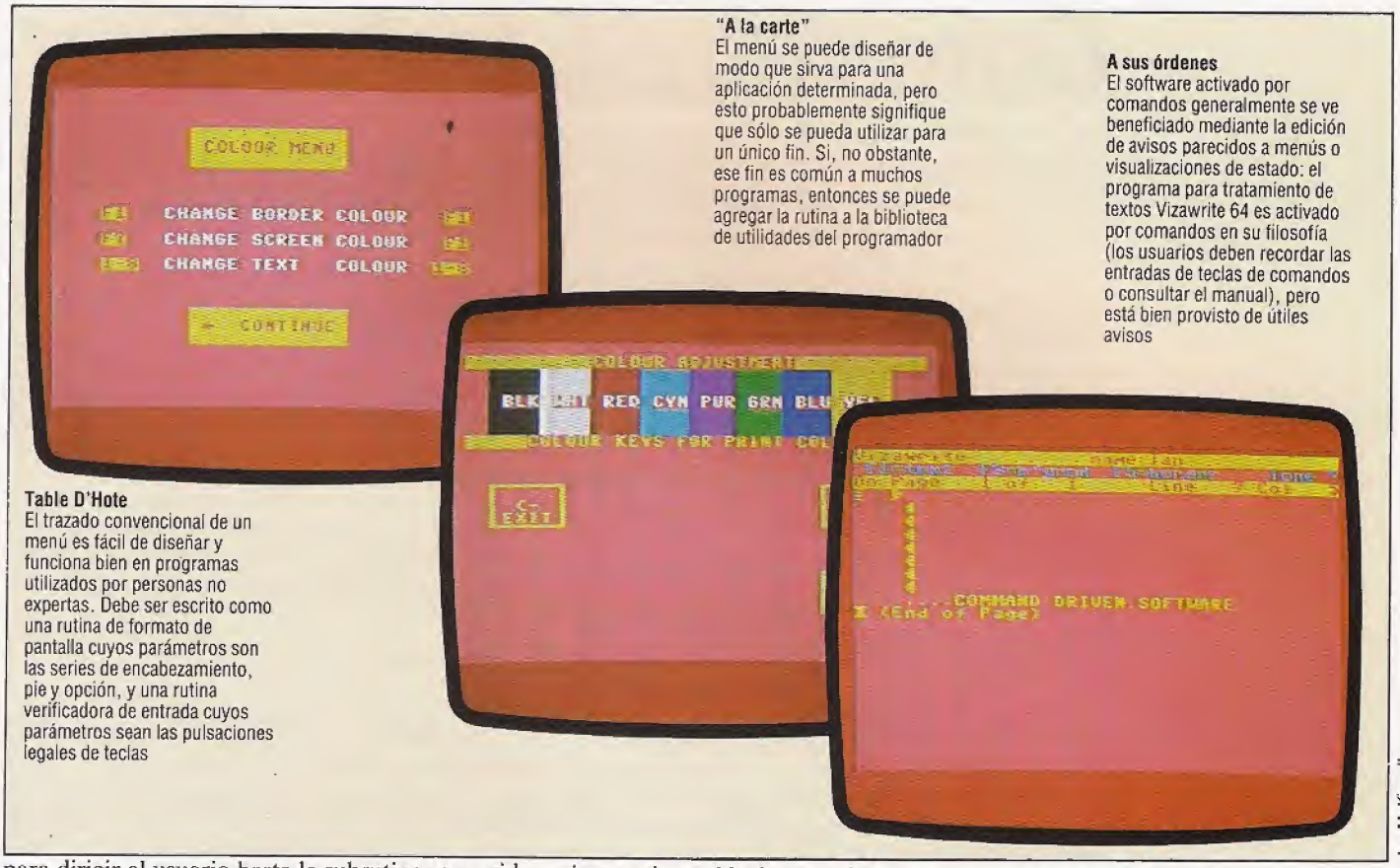


Table D'Hote
El trazado convencional de un menú es fácil de diseñar y funciona bien en programas utilizados por personas no expertas. Debe ser escrito como una rutina de formato de pantalla cuyos parámetros son las series de encabezamiento, pie y opción, y una rutina verificadora de entrada cuyos parámetros sean las pulsaciones legales de teclas

"A la carte"
El menú se puede diseñar de modo que sirva para una aplicación determinada, pero esto probablemente signifique que sólo se pueda utilizar para un único fin. Si, no obstante, ese fin es común a muchos programas, entonces se puede agregar la rutina a la biblioteca de utilidades del programador

A sus órdenes
El software activado por comandos generalmente se ve beneficiado mediante la edición de avisos parecidos a menús o visualizaciones de estado: el programa para tratamiento de textos Vizawrite 64 es activado por comandos en su filosofía (los usuarios deben recordar las entradas de teclas de comandos o consultar el manual), pero está bien provisto de útiles avisos

Ian McKinnell

Dos en tres
Por lo general existen muchas maneras de resolver el mismo problema; estas visualizaciones muestran distintas formas de permitirle al usuario modificar los colores de pantalla, borde y texto en un Commodore 64

para dirigir al usuario hasta la subrutina requerida. Este "intérprete de instrucciones" tiene cuatro tareas fundamentales. La primera consiste simplemente en esperar una entrada. La segunda, en "analizar gramaticalmente" esta entrada: el intérprete debe separar la línea de entrada en sus unidades funcionales. La tercera tarea consiste en interpretar la instrucción preparando la llamada a la subrutina apropiada. (¿Cuál es la dirección de la rutina? ¿Hay algún parámetro que pasar?) Por último, debe realmente llamar a la subrutina a ejecutar. Cuando el control regresa, el intérprete vuelve a retomar su primera tarea: ¡esperar!

El formato de una instrucción puede ser sumamente elaborado, y algunos lenguajes de instrucciones son similares a una forma simplificada de inglés. Un ejemplo de lenguaje de instrucciones es el esqueleto Unix, en el cual el formato típico de una instrucción es:

Instrucción+lista de parámetros opcionales
Ej.: L
o L−1

Aquí la instrucción Unix L lista un directorio de archivos, mientras que L−1 (donde −1 es un parámetro opcional) lista un directorio de archivos en formato "completo".

El analizador gramatical debe ser capaz de reconocer las diversas partes de la línea de instrucciones. El Unix mantiene las cosas sencillas (en la mayoría de los casos) tomando la primera palabra como la instrucción y reconociendo parámetros mediante un signo "menos" precedente. Los parámetros del lenguaje de instrucciones no son para que los utilice el propio intérprete de instrucciones, sino que los necesitan las subrutinas a las que llama el intérprete. Las rutinas empleadas en el sistema de instrucciones idealmente deberían adoptar un formato estándar para los parámetros de entrada. Si se hace esto, el intérprete de instrucciones puede pasar los parámetros en la forma en la cual fueron entrados (como series, tal vez).

Obviamente, es mucho más sencillo crear un intérprete de instrucciones que escribir un sistema de menús. Los usuarios experimentados tienden a preferir los sistemas de instrucciones o comandos, porque éstos son más rápidos y más flexibles que los programas activados por menú. La mayoría de los sistemas operativos son activados por comandos, lo que resulta lamentable para los usuarios novatos, dado que tales sistemas no proporcionan facilidades de señalización y las rutinas de ayuda en línea (en caso de que haya alguna) exigen cierto conocimiento del sistema. Además, la gran cantidad de instrucciones y de parámetros opcionales de un sistema de comandos típicos significa que incluso quienes estén razonablemente familiarizados con el sistema necesitarán facilidades de ayuda o bien consultar frecuentemente el manual de operatoria.

Los principiantes detestan las instrucciones y los expertos abominan los menús. Este problema es virtualmente insoluble, si bien existen algunos sistemas híbridos que pueden ser bastante eficaces. Por ejemplo, el programa para tratamiento de textos Wordstar es básicamente un sistema activado por comandos, pero al usuario puede parecerle un sistema de menús. Las instrucciones son códigos de control (algunos con parámetros) y el usuario ejecuta el sistema íntegramente con los mismos. Los menús que aparecen en la pantalla utilizan estas instrucciones como mnemónicas para seleccionar opciones de modo que, a medida que el novato va empleando los menús para ejecutar el programa, va aprendiendo al mismo tiempo las instrucciones.

Campos de trabajo

Tras el estudio de los registros, vamos a analizar los elementos de un programa en assembly y los tipos de direccionamiento

Una sentencia en assembly según la encuentra el ensamblador consta de tres partes, aunque no siempre estarán presentes todas. Estas tres partes, o *campos*, son:

- **El campo de etiquetas**, que ocupa la columna que queda más a la izquierda en el listado del programa en pantalla. Las etiquetas no son más que identificadores de números, relativos con frecuencia a direcciones de memoria. Consta la etiqueta de varios caracteres alfanuméricos (de uno a seis): el primer carácter debe ser alfabético, y el nombre en su conjunto no debe ser el mismo que el de un registro, o de un opcode en assembly o de otra etiqueta ya definida. Se trata de puros convencionalismos que otros ensambladores pueden cambiar. Si no se desea etiquetar una línea, al menos debe comenzarse con un carácter de espacio en blanco, para indicar un campo de etiqueta vacío. Del mismo modo, el campo de la etiqueta se concluye con un espacio en blanco; por ejemplo, LABLDA es sin duda una etiqueta correcta, mientras que LAB LDA se deberá interpretar como una etiqueta LAB seguida del opcode LDA.

El ensamblador se apoya en un *contador de posiciones*, que es un equivalente del registro contador de programa utilizado por el procesador. Retiene la dirección de la posición de memoria donde se almacenará el siguiente byte de la instrucción o del dato. En cuanto el ensamblador encuentra una etiqueta, guarda el identificador en un área de memoria llamada la *tabla de símbolos*, muy parecida a un vector en BASIC. Junto con el identificador almacena la dirección que indicaba el contador de posiciones en el momento de descubrir la etiqueta. Inmediatamente después de encontrar una etiqueta en assembly, el ensamblador coteja dicha etiqueta en la tabla de símbolos. Si está en la tabla, el ensamblador se limita a sustituirla por la dirección proporcionada, y si no la encuentra la almacena en la tabla junto con el contenido del contador de posiciones.

- **El campo de operador, instrucción u opcode**, que se sitúa a la derecha del campo de etiquetas. Es una expresión mnemotécnica de tres caracteres por lo general y un nombre de registro en casos necesarios. Por ejemplo, ADDA está compuesta de la expresión mnemotécnica ADD y el registro A. El opcode significa la operación a ejecutar por el procesador. Este campo acaba en un espacio en blanco, como el campo de etiquetas.

- **El campo de operandos o direcciones**, que proporciona información acerca de los datos sobre los cuales operará el opcode. Es frecuente que tales datos se den en forma de dirección, y todavía más frecuente que esta dirección se represente por una etiqueta.

Sea la sentencia en assembly:

```
LABEL1    ADDA    NUM1
```

que quiere decir: "introduce el dato almacenado en la dirección representada por el símbolo NUM1 en el acumulador A, sumándolo (ADD) con lo que éste contenga". La dirección de esta instrucción queda almacenada en una posición etiquetada por LABEL1 (*label*: etiqueta) si deseáramos dar un salto o bifurcación para volver a ella, sólo tendríamos que indicar este destino. NUM1 es una etiqueta especificada ya en otro lugar del programa, y significa la dirección donde está almacenado el dato. Veamos ahora este otro ejemplo:

```
          CLRA
```

que significa: "limpia (CLeaR) el acumulador A" (o sea, ponlo a cero). Se trata de un ejemplo de opcode sin operando. Obsérvese que esta línea no lleva etiqueta. Esto se debe a que las etiquetas son de uso opcional, se utilizan cuando se bifurca hacia ellas por medio de cualquier otra instrucción, o bien como una sentencia REM donde se etiquetan líneas significativas con anotaciones (REMarks) explicativas.

El empleo de etiquetas así resulta de gran ayuda, pero no pueden sustituir a los comentarios más amplios. Estas explicaciones se pueden añadir a cualquier línea dejando un espacio en blanco después del último carácter del operando, y agregando allí el comentario. Algunos ensambladores exigen que haya un carácter especial que les indique el inicio del comentario, y suele ser común comenzar una línea nueva de comentario con un asterisco.

El campo para operandos admite constantes que sean números o cadenas (variables tipo serie). Se suelen dejar los números en su forma decimal, a no ser que se indique por medio del prefijo $ o del sufijo H que están en hexadecimal (así, $AF08, AF08H); o bien se indique por el prefijo @ o el sufijo Q que están en octal (así, @6712,6712Q); o bien, en binario con el prefijo % o el sufijo B (así, %11010011, 11010011B). El código ASCII de un carácter puede servir de operando prefijándole un apóstrofo. Así, 'A significa 65 o $41.

Un valor particularmente útil es el contenido momentáneo del contador de posiciones. Éste no suele conocerse al introducir el programa, pero puede ser referenciado en el campo de operandos con un asterisco. Muchos ensambladores lo admiten en las expresiones aritméticas simples, por lo general la suma y la resta. Por ejemplo:

```
          LDA     *+5
```

quiere decir: "carga en el acumulador A el contenido de la posición de memoria cuya dirección es cinco bytes más alta que el contenido actual del contador de posiciones".

Directivas para el ensamblador

El ensamblador suele aceptar diversas *directivas* o *pseudo-ops*, escritas en el programa como opcodes cualesquiera. Ya hemos empleado dos de ellos anteriormente (véase p. 1019):

```
CR      FCB     13
```

FCB (*Fix Constant Byte*: fija un byte con valor constante) reserva un solo byte en la dirección indicada por el contador de posiciones, dándole el valor del operando. En este caso, la posición cuya dirección se representa con el símbolo CR queda inicializada con el número 13.

```
MEMTOP  FDB     $7FFF
```

FDB (*Fix Double Byte*: fija dos bytes) hace lo mismo, pero en dos bytes (16 bits). El espacio de memoria se puede reservar por bloques sin tener que fijar el valor de su contenido, con el empleo del pseudo-op RMB. Por ejemplo:

```
TABLE1  RMB     7
TABLE2  FCB     $F6
```

reserva siete bytes para una tabla de valores cuyo primer byte tiene la dirección que indica la etiqueta TABLE1. En la práctica, esto significa que si TABLE1 representa la dirección $C104, pongamos por caso, entonces TABLE2 representará una dirección que es siete bytes más alta, o sea, $C10B.

Toda una cadena de caracteres puede ser colocada en la memoria, como en este ejemplo:

```
ERRMSG  FCC     'ERROR
```

que inicializa cinco bytes de memoria por medio de los códigos ASCII correspondientes a las letras E,R,R,O,R. Es, pues, mucho más fácil introducir mensajes e indicaciones dirigidos al usuario en los programas en lenguaje assembly.

El pseudo-op ORG (origen) es muy importante. Especifica un valor para el contador de posiciones y sirve, al comienzo de un trozo de programa, para orientar al ensamblador sobre el lugar de la memoria donde comenzará a colocar el código del programa cuando se traduzca a código máquina. Siempre se deberían comenzar los programas con una sentencia ORG, aunque algunos ensambladores suplen su falta si no se hace. Se puede, y a veces se aconseja, especificar más de una sentencia ORG en el programa. Una directa ORG no requiere etiqueta ni operando.

La única sentencia que puede que se desee poner antes de ORG es EQU (EQU*als*: igual a), pues especifica los símbolos de las variables y no afecta al contador de posiciones. Por ejemplo:

```
RESET   EQU     $F100
```

significa que el símbolo RESET queda definido como representación del valor $F100. Luego RESET es como la abreviatura o sigla de un número, mientras que una etiqueta colocada al comienzo de la línea de sentencia representa la dirección donde se almacenará el dato o el código máquina.

Finalmente, la última directiva que debemos considerar es END, colocada al final del código fuente como un punto final para el ensamblador. Igual que ORG, no lleva ni etiqueta ni operando.

Tipos de direccionamiento del 6809

Una medida de la potencia de un lenguaje assembly la constituyen sus tipos de direccionamiento, es decir, el número de maneras de interpretar un operando. El 6809 admite muchos de estos tipos. Las instrucciones que hemos visto en los ejemplos se valieron o del modo *directo* o del *ampliado*, que significa que el valor o la etiqueta que se encuentra en el campo de operandos es la dirección de la posición de memoria que contiene el dato.

Direccionamiento directo significa que hay sólo un byte de dirección especificado para el operando de la instrucción. El procesador lo trata como el byte *lo* de la dirección completa del operando que consta de dos bytes, tomando por byte *hi* el contenido del *registro de página directo*, un registro de ocho bits de la CPU direccionable por el programa. Una ventaja de este modo es su flexibilidad y generalidad; por ejemplo, una subrutina puede ser escrita empleando el direccionamiento directo, no haciendo, por ello, referencia explícita a ninguna zona de memoria en concreto. Basta con especificar el contenido del registro de página directo antes de llamar a la rutina para señalar dónde se encuentran los datos en la memoria.

El direccionamiento ampliado consiste en especificar una dirección de dos bytes como operando de la instrucción. Esta instrucción hará siempre referencia a ese byte particular de la memoria, por lo que resulta una pieza inflexible de la codificación. El ensamblador reconoce ambos modos, el directo y el ampliado, por la naturaleza del operando.

Otro modo muy empleado es el *inmediato*, consistente en introducir el dato real en el mismo campo de operandos. Es un modo reconocible por el prefijo # delante del operando. Por ejemplo:

ORG $1000	la dirección inicial del lenguaje máquina es $1000
NUM1 FDB $FFFF	inicializa la posición NUM 1 con $FFFF
LDD NUM1	carga en el registro D el valor $FFFF contenido en NUM1
LDD #NUM1	carga en el registro D el valor $1000 de NUM1 que es el valor real o inmediato.

Se observará que la etiqueta NUM1 representa la dirección $1000. Es posible que usted piense que la instrucción ORG se colocará en la dirección $1000, y que la instrucción siguiente (junto, claro está, con su etiqueta NUM1) recibirán una dirección superior. Pero ha de recordar que ORG es una instrucción que dirige (directiva) el proceso de ensamblado, pero que no forma parte del mismo. Por tanto no ocupa espacio alguno de memoria, lo que explica que NUM1 tome el valor $1000 ya que éste es el valor del contador de posiciones en el momento de ser NUM1 tratado por primera vez por el ensamblador. Debe también observarse que LDD NUM1 carga el *contenido* de la posición NUM1 (que es $FFFF, especificado por la directiva FDB) en el acumulador D, mientras que LDD #NUM1 carga el *valor* propio de NUM1 (esto es, la dirección $1000 de la etiqueta).

Pequeños compromisos

En estos últimos tiempos Casio está intentando acceder al mercado especializado de ordenadores de mano y de bolsillo

Casio afirma controlar el 50 % del mercado internacional de calculadoras pero, sorprendentemente, la empresa cuenta con apenas 3 300 empleados en todo el mundo. En 1983 el volumen de movimiento total de Casio fue de 29 millones de dólares, una cifra baja para un fabricante de electrónica en grandes volúmenes. Tony Manton, director de ventas de calculadoras para Gran Bretaña, aclara que éste es un enfoque deliberado: "Somos una empresa muy pequeña y bastante conservadora. Las grandes campañas publicitarias no son nuestro estilo".

La empresa fue fundada por los cinco hermanos Kashio después de la segunda guerra mundial. En aquel entonces se llamaba Kashio Seisakujo y comenzó fabricando equipos para oficinas. A principios de los años cincuenta, la empresa desarrolló la 14-A Relay Calculator; ésta fue una de las primeras máquinas de calcular eléctricas y era tan grande como el tablero de un escritorio; pesaba 130 k. Se desarrollaron otras calculadoras y en 1957 la empresa cambió su nombre por el de Casio Computer Company Ltd.

Casio se estableció en Gran Bretaña en 1974 y se halló con un mercado fácil para calculadoras electrónicas de precio reducido. En 1982 se lanzó el primer ordenador de bolsillo Casio, el FX-720P. Éste se diseñó para el usuario científico y la disposición de su teclado era inusual, con las teclas dispuestas por orden alfabético en vez de en el tradicional formato QWERTY. Las ventas fueron decepcionantes y la máquina se sustituyó por la FX-700P, que estaba equipada con un teclado QWERTY.

A ella siguió el PB-100, un ordenador tamaño de bolsillo dirigido al usuario de gestión. Similar en diseño al FX-702P, incorporaba una visualización en cristal líquido y un teclado numérico. Casi produjo, asimismo, grabadoras de cassette, impresoras-plotter y paquetes de RAM construidos a la medida para ambas máquinas.

Un reciente sustituto del PB-100 es el FX-750P. Este ordenador configura dos ranuras "RAM-card", que alojan pequeños paquetes metálicos (del tamaño aproximado de una caja de fósforos) y pueden retener hasta 4 Kbytes de datos. Cada ficha está equipada con una pila de tres voltios que almacena el contenido de memoria después que se saca la ficha del ordenador, lo que permite, por lo tanto, guardar y cargar programas en la máquina cuando sea necesario. Cada pila proporciona un almacenamiento de un año; cuando es necesario sustituir la pila, los programas se vuelven a cargar desde cinta.

Casio también produce el FP-200, un ordenador de mano que está equipado con 8 Kbytes de RAM. Ésta se puede ampliar a 32 Kbytes. El FP-200 posee una visualización en cristal líquido que ofrece ocho filas de 20 caracteres en modalidad de textos y tiene una resolución de gráficos de 160×64 pixels. La máquina más nueva de la gama Casio es la SL-800. Se trata de una calculadora de precio reducido que tiene aproximadamente el mismo tamaño y peso que una tarjeta de crédito. El delgado diseño es consecuencia de un proceso de fabricación que se denomina *filmación*, en el cual los componentes se imprimen en película laminada en vez de ser soldados a una placa de circuito impreso.

En muchos países europeos y en el Lejano Oriente, Casio distribuye ordenadores de gestión y ordenadores personales estándar MSX. Sin embargo, en Gran Bretaña la empresa se ha concentrado en calculadoras y pequeños ordenadores. Preguntado acerca de las razones por las que Casio no ha intentado penetrar en el mercado británico personal y de gestión, Tony Manton alude a la naturaleza despiadada de este sector de la industria: "Nuestra intención es ir expandiéndonos hacia arriba lentamente. Es necesario que eduquemos a las personas para que sepan que los ordenadores manuales y de bolsillo son algo más que calculadoras."

Además de cualquier posible expansión en este campo, Casio está aplicando la experiencia adquirida en el campo del teclado electrónico para producir una serie de máquinas alrededor de la interface digital MIDI (véase p. 1014). Dichas máquinas habrán de hacer su presentación en el mercado en un futuro muy inmediato.

Centro de Investigación y Desarrollo de Casio, en Tokio

En la copa del árbol
Tadao Kashio, presidente de Casio, es uno de los cinco miembros de la familia Kashio situados en la cúspide de la estructura corporativa de la firma

Calidad de sonido

Marcus Wilson-Smith

La MIDI ofrece al usuario de un ordenador personal una forma de entrar en el extraño y fascinante mundo de la música electrónica

En el campo de la educación, la MIDI ofrece algunos verdaderos adelantos, tanto en la escuela como en el hogar. Aprender a leer música suele ser difícil, incluso cuando el estudiante toca ya bastante bien. Las primeras etapas en el aprendizaje de cosas tales como sostenidos y bemoles e indicaciones de compás pueden ser arduas y llevar mucho tiempo, y la relación entre la partitura y lo que en realidad se escucha muy raramente resulta obvia.

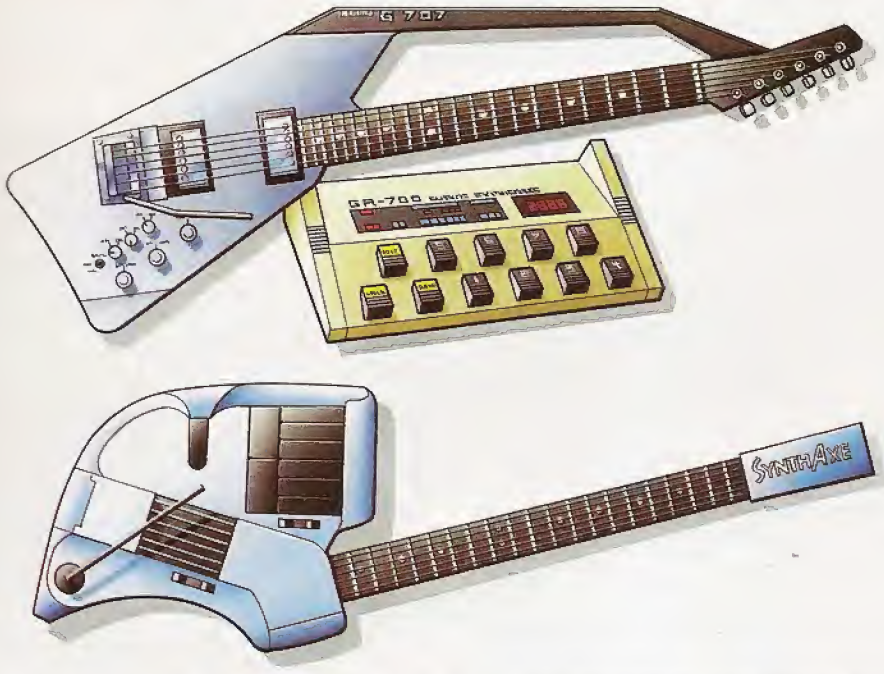
Uno de los principales aspectos del problema radica en la naturaleza de la música en sí: para que tenga algún sentido, deben coordinarse en el tiempo una serie de eventos. Si la única forma de que un principiante pueda seguir la notación gráfica es ir interrumpiendo la música, entonces todos los indicadores de duración de tiempo del pentagrama pierden todo sentido. De modo similar, el principiante suele tardar algún tiempo tratando de interpretar un compás o frase determinados; mientras realiza esto la música habrá pasado a ser otra cosa totalmente diferente.

La MIDI elimina estos obstáculos. Puede permitir almacenar la ejecución de un sintetizador en la memoria de un micro y, con el software apropiado, proporcionará una visualización gráfica de la música que se esté tocando. Ésta es una valiosa ayuda para cualquier estudiante de música. Significa que, por ejemplo, si se toca un *do* normal en el teclado del sintetizador, se mostrará un *do* normal en el pentagrama de la visualización en pantalla. Si se mantiene un acorde en *re menor* durante un cierto tiempo, se visualizarán los componentes armónicos del acorde (*re, fa, la*), junto con la duración adecuada.

Esta idea se puede ampliar mediante un software que ejecute una pieza de música preprogramada en el sintetizador conectado en interface mientras en la pantalla se va desplazando un pentagrama con la notación completa. En esta situación, tanto la música como su notación se pueden detener simultáneamente y volverse a ejecutar a partir de un número de compás especificado si el usuario tropieza con un problema. Además, se puede modificar el sonido global de la música cambiando los parámetros de control del sintetizador, introduciendo, por lo tanto, al usuario en el arte del arreglo.

Una vez adquirido un grado de confianza en cuanto a lectura de música, entonces será más fácil escribirla con la ayuda del teclado alfanumérico de un ordenador. Esto puede implicar la entrada de datos de ejecución sin ninguna referencia inmediata a un sonido del sintetizador, y comparar después el resultado con las intenciones originales. Una vez adquirida esta destreza más avanzada, se puede abandonar la notación en pentagrama en favor de otro sistema, como el MCL (*Music Composition Language:* lenguaje para composición musical). Para la música electrónica, el MCL es un medio más apropiado para entrar los datos, dado que incluye una especificación de características que son exclusivas de la producción de sonido electrónico. Para la aplicación MCL no se ha desarrollado ningún estándar: cada máquina posee su propio MCL. La notación de pentagrama, si bien constituye una guía visual inmediata, es un sistema cuya existencia data de varios siglos antes de la electrónica, y no se puede adaptar a todas las nuevas notaciones que requiere la música electrónica.

Sonidos nuevos, estilo nuevo
Uno de los numerosos nuevos sintetizadores de la gama DX de Yamaha, el DX7, incorpora un método para construir sonido, la síntesis FM, que previamente estaba restringido a máquinas muy caras. En vez de tomar un sonido existente y modificarlo haciéndolo pasar a través de filtros o adaptando los valores de envoltura, el DX7 crea sus propios sonidos complejos mediante la combinación de seis formas de onda de diversas maneras. Por consiguiente, el DX7 se aproxima a los sonidos de instrumentos acústicos mucho más que los otros sintetizadores. También incorpora control de respiración, de modo que un músico puede soplar en un receptor y agregarles a los sonidos del saxofón o la trompeta, por ejemplo, variaciones tipo respiración. El DX7 puede utilizar paquetes de RAM con características de sonido pregrabadas, o almacenar sonidos creados por el usuario en paquetes de RAM



Kevin Jones

Sonido sintetizado

Recientemente varios fabricantes han desarrollado sintetizadores que se activan desde las cuerdas de una guitarra en vez de desde las teclas de un piano. El Roland GR700 utiliza cuerdas de guitarra estándar. La compleja información del sonido se recoge mediante una entrada hexafónica (seis sonidos) y se envía al sintetizador, donde se le agregan parámetros ajenos a la guitarra. Un original enfoque a la misma técnica lo constituye el SynthAxe, un sintetizador controlado por 6809. El SynthAxe recoge datos de sonido desde una conexión eléctrica entre la cuerda y el traste. Unos sensores en la parte superior del clavijero captan las variaciones de inclinación y resbalamiento. Para mejorar aún más el sonido se utilizan un segundo grupo de cuerdas y un pequeño teclado

Para muchos usuarios de microordenadores, entender la notación de pentagrama o el MCL será la clave para aprovechar la MIDI al máximo. En cuanto a los estudiantes secundarios y universitarios, un obstáculo fundamental que deben salvar es el carácter que posee actualmente la educación musical. Incluso la mayoría de los estudiantes de música se ven abocados a un área de estudios firmemente asentada en la música clásica europea. La mayoría de los músicos clásicos y los profesores identifican la música electrónica con los compositores más vanguardistas o radicales de las dos últimas décadas y, hasta cierto punto, con la música *pop* contemporánea. Ninguno de estos campos se acepta en pie de igualdad con la música clásica. De hecho, algunos puristas llegan incluso a cuestionar que se trate siquiera de "música".

Parece improbable, por consiguiente, que el potencial de la MIDI en el área educativa se explore demasiado en la vertiente musical de la tendencia principal, especialmente porque, además de la electrónica musical, se requiere experiencia con ordenadores. Podrían existir algunos cursos de informática con facilidades para experimentación acústica que fueran adecuados para una clase de músicos de sintetizador MIDI. Si éste resultara ser popular en un medio de tales características, se requeriría una solución simple, tal como el empleo de auriculares. Pero, dado que la mayoría de las unidades MIDI están diseñadas para conectarse en interface con uno o más sintetizadores, para tratar con esto sería preciso hacer uso de varias señales. Incluso a un nivel básico, el empleo de la MIDI en educación implica el desarrollo de facilidades de música con ordenador en un estudio, y esto parece exigir una participación activa por parte de estudiantes de informática además de estudiantes de música.

Para las actuaciones en vivo, la MIDI es fundamentalmente un medio de integrar un cierto número de sintetizadores, secuenciadores y máquinas de ritmos en un único sistema controlable. La posibilidad que más preocupa a los músicos que utilizan una gran proporción de secuenciación en sus actuaciones es una pérdida de sincronización entre las unidades, y el resultante fracaso musical. Se ha sabido que músicos como los Thompson Twins y Howard Jones tocan cintas de apoyo grabadas en el estudio mientras "actúan en vivo", para no correr riesgos. Al menos en teoría los grupos de multisintetizadores deberían tener, gracias a la MIDI, actuaciones más libres de problemas.

Una característica de este avance es que tales grupos ya no tendrán apariencia de ser de "multisintetizadores". Desde principios de los años setenta, por lo general se ha pensado en el sintetizador como un instrumento de teclado, con gran número de mandos de control de parámetros y guías deslizadoras dispuestas en una tira encima de las teclas. Pero si un sintetizador de teclado está utilizando la MIDI para controlar a un segundo o un tercero, entonces ya no existe necesidad alguna de tener más teclados que el del instrumento maestro. A medida que el empleo de la MIDI se va generalizando, están saliendo al mercado más "módulos de sintetizador". Éstos son, simplemente, las unidades de generación de sonido y de secuenciación que previamente formaban parte de los instrumentos de teclado. Dado que estos módulos poseen poco o ningún interés *visual*, no hay necesidad de exhibirlos en el escenario.

Existe otro desarrollo que antecede a la MIDI, pero que es probable que sea objeto de más atención a medida que vaya disminuyendo la cantidad de teclados. Este desarrollo es la posibilidad de controlar la síntesis de sonido electrónico mediante datos de ejecución en cuerdas producidos con guitarras, y mediante datos de control de respiración y boca provenientes de instrumentos de viento. En comparación con la simple acción mecánica de pulsar una nota en un teclado, puntear una cuerda o hacer vibrar una lengüeta implica que al instrumento en cuestión se le está transmitiendo una información acústica más compleja. Si esta información se codifica digitalmente y se transmite

vía MIDI a un módulo de sintetizador fuera del escenario, no existe ninguna razón por la cual un saxofonista no pueda ser el principal controlador de la electrónica de un grupo, incluso de su máquina de ritmos. Yamaha ha incorporado control de respiración en su sintetizador DX7, y se ha diseñado la SynthAxe (una guitarra digital desarrollada recientemente) para emplear la MIDI con la misión de controlar la salida de un Fairlight.

Ello significa que se podría producir en el Yamaha un sonido de cuerda con las características de ejecución exclusivas de un saxofón y, de igual modo, se podría articular una muestra de trombón Fairlight mediante el rasgueado de una guitarra. Si bien ninguno de estos desarrollos es inminente (al fin y al cabo, el SynthAxe es una "guitarra" muy cara), constituyen indicios de probables tendencias para las actuaciones en vivo del futuro cercano. Es probable que se produzca una disminución gradual de los teclados en escena; adquirirán mayor importancia los instrumentos de cuerdas, viento y posiblemente de percusión afinada, como vibráfonos, y en la medida en que la tecnología del muestreo de sonidos se vuelva más asequible al usuario, podría llegar a predominar el sonido acústico.

Para finales de los ochenta, los tradicionalistas de la música puede que se sientan desagraviados al ver nuevamente una unidad estilo jazz de guitarra, saxofón, contrabajo y batería. Puede que, sin embargo, se sientan confundidos por el hecho de que el guitarrista esté tocando un vibráfono invisible y el saxo esté emitiendo sonidos de batería.

Para muchos grupos habituados a las actuaciones en vivo, la primera experiencia en un estudio de grabación avanzado puede ser intimidante. Se hallan frente a instrumentos musicales y sistemas operativos con los que nunca se habían encontrado antes y se les asigna un productor que tal vez no conozca su trabajo ni sus intenciones con particular profundidad. Y su compañía discográfica espera que produzcan versiones "mejores y más espectaculares" de sus éxitos en el escenario en este entorno tan poco familiar. El hecho se puede comparar con una compañía teatral semiprofesional a la que se colocara en el plató de una película de enorme presupuesto esperando un éxito de taquilla instantáneo. En algunas ocasiones la transición tiene éxito y todos los implicados quedan satisfechos. Pero con mucha frecuencia las ideas originales se pierden en un laberinto de dispositivos de estudio y al final esta magna empresa se ve abocada a un oneroso fracaso. Lo más frecuente es que éste se produzca cuando el grupo descarta su propio equipo, ya familiar (y, en realidad, su propio "sonido"), y se vuelca en los más deseables instrumentos que hay disponibles en el estudio. Pero puede que una idea que funcionaba bien en un sintetizador Mini-Moog se malogre por completo al ejecutarse en un ordenador de muestreo digital Fairlight, y si este decepcionante resultado se produce con suficiente frecuencia, entonces la justificación para el empleo de tal estudio comienza a perder fuerza.

Sin embargo, si los músicos del grupo ya conocen la MIDI y si han utilizado un microordenador para almacenar secuencias y otros datos de control musical, ya estarán familiarizados con los procedimientos que se siguen en los estudios avanzados. En el nivel más inmediato, sería posible probar ideas empleando una sucesión de distintos instrumentos del estudio con un mínimo de dificultad, y con el conocimiento previo de que las ideas y las frases se pueden transformar simplemente mediante el cambio de sintetizadores.

El conocimiento de la MIDI también resulta valioso cuando se trata de familiarizarse con sistemas de estudio distintos de aquellos directamente implicados en la generación de sonido. Una mesa mezcladora de lógica transistorizada, por ejemplo, posee un ordenador exclusivo que recordará y volverá a ejecutar cualquier serie de decisiones efectuadas en las etapas finales de la grabación, lo que se conoce como mezclado. Cuando se ha grabado toda la música en 24 pistas de cinta separadas (la guitarra en una pista, las vocales de apoyo en otra, las vocales principales en otras tres, etc.), empieza la crucial tarea de equilibrar y mezclar todos los elementos. Generalmente es en este punto donde las partes individuales son tratadas con cualquier "efecto" requerido para hacer que destaquen o se fundan en la mezcla. Una única nota de trompeta puede necesitar el agregado de reverberación en un solo punto y, con otras 23 cosas sucediendo al mismo tiempo, es fácil equivocarse. Utilizar un ordenador para tratar y controlar tales incidentes durante el mezclado es como la secuenciación MIDI a gran escala.

Otra técnica, desarrollada originalmente para la sincronización y edición de video, pero que está adquiriendo protagonismo en la producción de música, es el empleo de código de tiempo. El código de tiempo es como un reloj digital más una señal de disparo, pero grabado en cinta. Utiliza palabras de 80 bits para proporcionar datos de sincronización cuando se graba música con secuencias de video y permite secuenciar juntos eventos musicales y montajes de video de fracciones de segundo.

Los músicos, en consecuencia, tienen fundadas razones para adquirir instrumentos compatibles con la MIDI; pero, además, ésta es una buena introducción a los sistemas musicales más avanzados.

Creatividad a gran escala
Laurie Anderson, poetisa y artista neoyorquina, realiza una insólita mezcla de sonidos y equipo de sonido con tecnología de cine, de cinta y de video, para crear un estilo exclusivo. En canciones como *O Superman* y *Mr Heartbreak* utiliza o es secundada por los más disímiles objetos, desde campanas africanas a instrumentos electrónicos ultramodernos como el Vocoder y el Synclavier

Método eficaz

En este capítulo estudiaremos la utilización de procedimientos para definir otros procedimientos

A fin de ilustrar este principio de emplear procedimientos para definir datos, tomemos como ejemplo el puzzle tangram. Se trata de un cuadrado que ha sido dividido en siete trozos geométricos, que se combinan de diversas maneras para formar distintas formas. En nuestro ejemplo utilizaremos las siete piezas básicas para crear una forma que se parezca a un perro. Comenzamos por definir procedimientos LOGO para cada una de las piezas básicas; estos "procedimientos de piezas" se incorporan luego a otro procedimiento, al que se le da el nombre de PERRO. Dado que la tortuga debe estar correctamente posicionada antes de que se dibuje cada pieza, también se deben utilizar otros procedimientos (desde MOVER1 a MOVER7).

Sería igualmente sencillo realizar este dibujo simplemente encadenando las instrucciones una tras otra en un procedimiento largo. Nuestro método utiliza los principios del diseño "de arriba abajo", o *top-down* (véase p. 956). En líneas muy generales, este método significa simplemente descomponer un problema en un número de partes y después proceder a resolver cada una de ellas individualmente. La gran ventaja de este enfoque es que el programador de LOGO puede definir un procedimiento que contenga subprocedimientos que aún no se han definido. El procedimiento principal no se puede ejecutar, por supuesto, hasta que se escriban los subprocedimientos o éstos sean sustituidos por rutinas ficticias. Para mostrarle cómo funciona esto, consideremos cómo se construyó el programa que dibuja al perro.

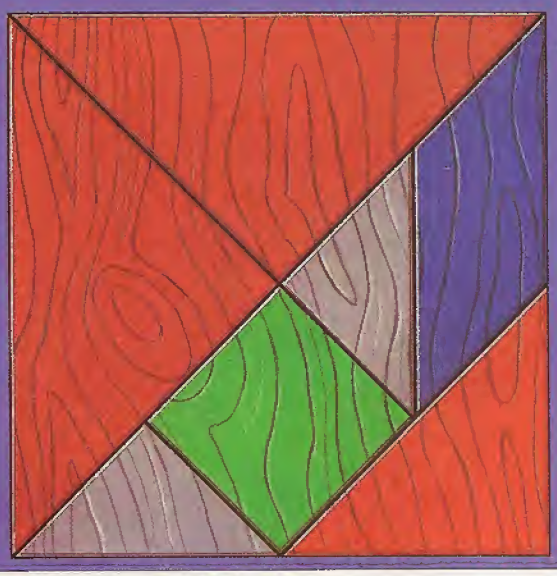
Primero se escribió el procedimiento PERRO, aun cuando todavía no existía ninguno de los procedi-

Dando forma

El puzzle tangram es un conjunto de siete formas que se pueden disponer en diversas combinaciones para crear diseños sencillos. Aquí listamos los procedimientos LOGO para dibujar las siete piezas básicas del tangram, así como un programa de demostración que dibuja la figura de un perro. El procedimiento PERRO empieza dibujando con el triángulo para la pata trasera del animal

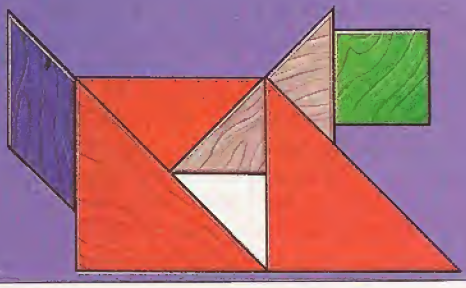
Procedimientos tangram

```
TO CUADRADO
   REPEAT 4 [FD 25 RT 90]
END
TO PAR
   REPEAT 2 [FD 25 RT 45 FD 35 RT 135]
END
TO TRI1
   FD 25 RT 135 FD 35 RT 135 FD 25 RT 90
END
(Hay dos triángulos de este tamaño)
TO TRI2
   FD 35 RT 135 FD 50 RT 135 FD 35 RT 90
END
TO TRI3
   FD 50 RT 135 FD 71 RT 135 FD 50 RT 90
END
(De este tamaño también hay dos triángulos)
```



Programa Perro

```
TO PERRO
   TRI3 MOVER1 PAR MOVER2 TRI2 MOVER3
   TRI1 MOVER4 TRI3 MOVER5 TRI1 MOVER6
   CUADRADO MOVER7
END
TO MOVER1
   PU FD 15 LT 45 PD
END
TO MOVER2
   PU RT 45 FD 35 LT 45 BK 35 PD
END
TO MOVER3
   PU LT 45 BK 25 PD
END
TO MOVER4
   PU RT 90 BK 25 PD
END
TO MOVER5
   PU FD 50 RT 45 PD
END
TO MOVER6
   PU FD 25 RT 135 FD 5 LT 90 PD
END
TO MOVER7
   PU LT 90 FD 5 RT 45 BK 25 RT 45
   BK 50 LT 90 BK 50 PD
END
```



Kevin Jones

mientos que lo componían. Después escribimos por separado cada uno de los procedimientos de dibujo de formas. A los mismos les siguieron los procedimientos de posicionamiento. Cada vez que se escribía un nuevo procedimiento, se ejecutaba PERRO para comprobar que todo se fuera acoplando correctamente. Cuando el LOGO llegaba a un procedimiento MOVER que aún no se hubiera escrito, se interrumpía con un mensaje de error. No obstante, era fácil decir a partir del dibujo si hasta ese momento todo era correcto o si existía algún error en el último procedimiento MOVER.

Nuestro conjunto de procedimientos demuestra otro punto importante: cada uno de los procedimientos de forma, y el propio procedimiento PERRO, deja el estado de la tortuga sin ninguna alteración. Es decir, al final del procedimiento la tortuga está en la misma posición y con el mismo encabezamiento que antes de que se ejecutara el procedimiento. Se dice que tales procedimientos son *transparentes*. Hacer que los procedimientos sean transparentes facilita la tarea de juntar procedimientos para construir dibujos más complejos. Tomemos el procedimiento PERRO, por ejemplo: una vez posicionada la tortuga sabemos que después de dibujar una pieza retornará a la posición en la que estaba cuando la empezó. De modo que no es necesario que sepamos nada acerca del funcionamiento interno de los procedimientos al objeto de colocar las piezas juntas. Haciendo que PERRO sea transparente conseguimos que sea más fácil utilizar este procedimiento como parte de otro; por ejemplo, podríamos dibujar una pantalla llena de perros.

Espacio de trabajo del LOGO

Ahora ya tendrá en la memoria del ordenador cierta cantidad de procedimientos, de modo que analicemos en mayor profundidad la organización de la memoria del LOGO. La memoria de trabajo se compone de una lista de *nodos* (cada uno de ellos de cinco bytes). Una vez cargado el LOGO, se dispone de 1 000 a 3 000 de éstos, según la máquina que emplee. A medida que se van definiendo procedimientos, estos nodos se consumen. También se utilizan más nodos cuando se ejecutan procedimientos o cuando se utilizan variables (que analizaremos más adelante en el curso).

Los procedimientos que se han definido constituyen el *espacio de trabajo*. Se puede ver qué procedimientos están retenidos en el espacio de trabajo entrando el comando POTS (por PRINTOUT TITLES: imprimir títulos). Para ver un procedimiento en particular, emplee PO (por PRINTOUT: imprimir); por ejemplo, PO CUADRADO. Si un procedimiento ya no se necesita más, se puede liberar el espacio de trabajo que ocupa mediante la utilización de ERASE: la instrucción o comando ERASE CUADRADO eliminaría de la memoria el procedimiento llamado CUADRADO. Borrar un procedimiento libera los nodos que éste utiliza. El LOGO marcará estos nodos pero no los agregará todavía a la lista de nodos libres; en cambio, seguirá trabajando con su lista actual de nodos libres hasta que se consuman todos éstos. Entonces repasará su memoria, reuniendo todos los nodos que se hayan liberado y los utilizará para formar una nueva lista de nodos libres. A este proceso se alude como *recolección de basura (garbage collection)* y es el motivo por el cual el LOGO parece dudar uno o dos segundos de vez en cuando.

Cómo salvar procedimientos

Con el fin de almacenar permanentemente los procedimientos en disco, usted debe salvar su espacio de trabajo como un archivo. Utilizando MISPROCS como ejemplo de nombre de un archivo, debería entrarse el comando SAVE "MISPROCS (observe las comillas: solamente abren, no cierran). El espacio de trabajo propiamente dicho no sufrirá ninguna alteración por esto. El archivo se podría cargar con READ "MISPROCS". Esto hace que se definan los procedimientos que contiene el archivo y se agreguen al espacio de trabajo en curso. Si se define un procedimiento con el mismo nombre que otro ya retenido en el espacio de trabajo, entonces la nueva definición reemplaza a la anterior.

Otras instrucciones o comandos útiles para la manipulación de disco son CATALOG y ERASEFILE. CATALOG (catálogo) da una lista de todos los archivos del disco, y ERASEFILE "MISPROCS" borraría el archivo MISPROCS del disco. Las versiones de LOGO basadas en cassette usan instrucciones diferentes.

Abreviaturas

ERASE	ER
PRINTOUT	PO
PRINTOUT TITLES	POTS

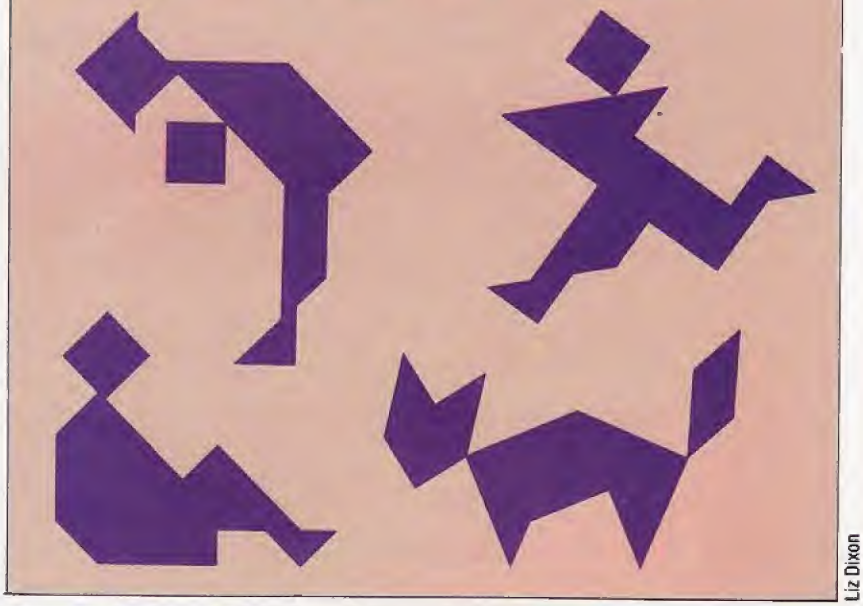
Problemas con procedimientos

1) Escribir procedimientos para las otras formas tangram ilustradas. (Primero tendrá que resolver el puzzle de cómo construir la forma a partir de las distintas piezas, ¡por supuesto!)

2) Escribir un procedimiento para dibujar una "casa" (un triángulo equilátero encima de un cuadrado)

3) Escribir un procedimiento para dibujar un tablero de cuadrados de cinco por cinco

4) Reescribir el procedimiento empleado anteriormente para dibujar una estrella de seis puntas de modo que utilice subprocedimientos



Liz Dixon

Complementos al LOGO

En todas las versiones LCSI, los nombres de los procedimientos deben ir precedidos por comillas si son argumentos de PO o ERASE: por ejemplo, PO "CUADRADO y ERASE "CUADRADO.

Para leer un archivo en disco use LOAD "MISPROCS.

Las versiones basadas en cintas de cassette o microdrives poseen instrucciones algo diferentes a aquellas basadas en disco. Consulte el manual.

Biblioteca pública

La creación de bibliotecas o librerías de rutinas de uso general permite optimizar los esfuerzos de programación

Seguir los métodos de diseño estructurado que ya hemos descrito en este curso puede parecer un enfoque algo largo; pero, en realidad, ahorra tiempo (no sólo en la codificación sino especialmente en la depuración de un programa). Ello se debe a que los programas que se crean directamente desde el teclado tienden a tener estructuras y algoritmos innecesariamente complicados. Esto significa que se tarda más tiempo en escribirlos, son más susceptibles a la aparición de errores y, dado que son más difíciles de seguir, requieren mucho más esfuerzo para su comprobación y depuración. Planificar el programa por adelantado simplifica la estructura y los algoritmos y, por consiguiente, conduce a menos errores de codificación y una comprobación y depuración más sencillas.

Aún más importante, diseñar por anticipado le ahorra al programador el escribir una estructura de control o de archivo que posteriormente pueda resultar inadecuada (quizá por no haber destinado espacio suficiente para un campo del archivo). Los problemas como éste, que son fundamentales para la forma en que opera el programa, pueden determinar que se hayan de reescribir porciones muy importantes del mismo.

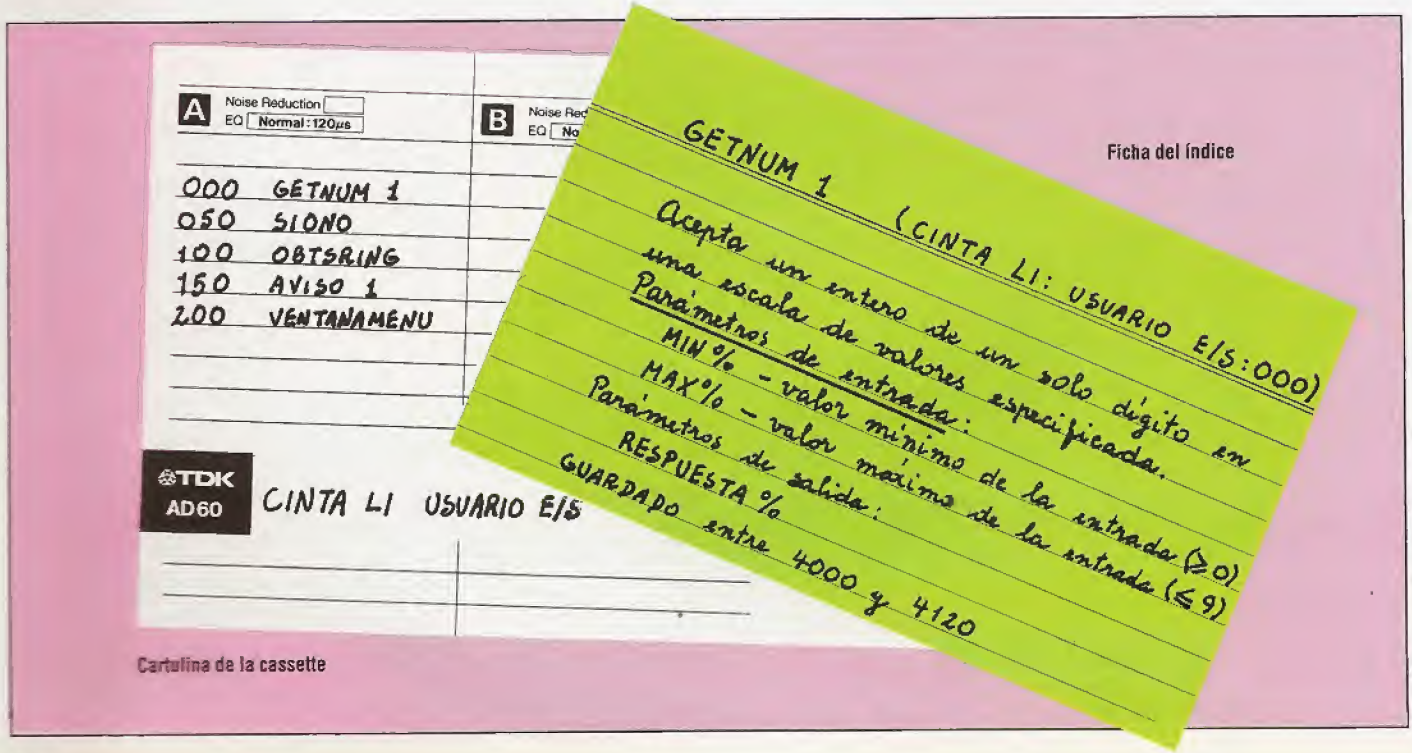
Quienes posean un teclado "apropiado" tipo máquina de escribir tal vez deseen invertir algo de tiempo aprendiendo mecanografía al tacto. Aparte de esto, sin embargo, es poco lo que se puede hacer para aumentar la velocidad a la cual se escriben en el teclado las líneas del programa. No obstante, el proceso de codificar programas se puede acelerar de diversas maneras. La primera es la más sencilla: definir, adoptar y utilizar cierto número de "convenciones" cuando se codifica. Tales medidas incluyen: hacer uso de determinados tipos de nombres para variables locales, para diferenciarlos de las variables del programa principal o variables globales; empezar cada subrutina en líneas que acaben en 000; acabar cada subrutina con el RETURN en una línea propia; empezar cada tipo de subrutina en un bloque determinado de líneas (rutinas para tratamiento de archivos entre la 9000 y la 9999, utilidades, de la 50000 en adelante, etc.).

La utilización de estas convenciones reporta numerosos beneficios: uno no tiene que ir a la pesca de las rutinas del menú porque sabe que siempre están en el mismo sitio, ni tiene que preocuparse de si ya ha empleado o no el mismo nombre de variable en el programa principal y en una subrutina, porque su nombre indicará que se trata de una variable local.

Sistema uniforme
Las bibliotecas de subrutinas no tienen ninguna utilidad si carecen de un sistema de documentación homogéneo. Esto es particularmente cierto en el caso de los usuarios de cassettes: inspeccionar el contenido de una cassette indocumentada cargando y listando cada programa es una tarea improba

Bibliotecas de programas

Tales técnicas de codificación son asimismo útiles cuando se crean bibliotecas de programas. Una biblioteca de subrutinas bien organizada puede significar un ahorro de tiempo de codificación para un programa largo de hasta el 50 %. La mejor forma de comenzar una biblioteca de este tipo es remitirse a los programas existentes y extraer todas las subru-



Ficha del índice

Cartulina de la cassette

Mike Cloves

tinas que estén bien escritas y posean alguna aplicabilidad general (rutinas de E/S, rutinas de verificación de datos, conversión de mayúsculas a minúsculas, etc.). Cada rutina se debe guardar como un archivo separado y éstos se deben agrupar entre sí de acuerdo a su función (si se van a almacenar en cinta entonces cada grupo de funciones se debe almacenar en una cassette separada), con nombres de archivo que tengan algún sentido para identificarlos. Lleve un índice de fichas o una base de datos de los nombres de los archivos, junto con una descripción de qué hace cada rutina.

Huelga decir lo importante que es asegurarse de que todas las rutinas de la biblioteca estén rigurosamente comprobadas y depuradas. Las mismas se habrán de utilizar en programas para los cuales no fueron diseñadas especialmente, de modo que asegúrese de que detectarán cualquier valor de entrada ilegal. Asimismo, debe asegurarse de que ninguna salida de valores de las rutinas de la biblioteca suponga problemas para el programa que las emplea. Haga a cada rutina tan eficaz como sea posible e incluya toda la documentación interna que sea necesaria para que se entienda la función de la rutina cuando sea utilizada en el futuro. Agregue una rutina al grupo cuando surja la necesidad: no tiene ningún sentido agregar nuevas rutinas demasiado específicas porque la experiencia demuestra que esto es un esfuerzo en gran medida desperdiciado. No se olvide de numerar las líneas de las rutinas de la biblioteca según la convención establecida (ello le evitará la RENUMeración cuando las rutinas se fusionen en un nuevo programa). En las revistas de informática, que con frecuencia publican listados de rutinas así como programas completos, se pueden encontrar eficaces rutinas de biblioteca.

Para hacer uso de una biblioteca como ésta es necesario disponer de una forma de fusionar rutinas entre sí para formar un programa completo. A quienes utilizan lenguajes compilados se les suele proporcionar un programa "montado" *(link-editor)* o similar; éste toma módulos compilados y los une para hacer un programa ejecutable. Para los programadores de BASIC, a menos que dispongan de un compilador, la forma más sencilla de conseguir esto consiste en emplear una combinación de instrucciones RENUM (renumerar) y MERGE (fusionar). Para fusionar una rutina de biblioteca en un nuevo programa, primero cargue el programa, decida dónde irá la rutina de biblioteca y asegúrese de que haya un bloque de números de línea sin utilizar suficiente como para darle cabida. De ser necesario, RENUMere la rutina de biblioteca de modo que encaje en el espacio que tiene asignado. Luego emplee la instrucción MERGE para juntar los dos programas; compruebe que todo funcione correctamente y guarde (SAVE) el nuevo programa con la rutina de biblioteca en su sitio.

Esfuerzos en grupo

Con frecuencia se da el caso de que los usuarios de ordenadores personales trabajan en grupo para escribir programas, ya sea en la escuela o en sus clubes de usuarios. La mayor parte de cuanto se ha dicho acerca del diseño de programas y de la eficacia del programador cobra especial relevancia para tales esfuerzos de equipo. En realidad, la mayoría de estas ideas y el concepto de programación estructurada se desarrollaron con el fin de dividir el peso del trabajo en los proyectos de programación comerciales. Por consiguiente, distintos programadores podrían trabajar en diferentes partes del mismo programa al mismo tiempo para producir un programa operativo. Para que los programadores de BASIC trabajen de este modo, es esencial estar de acuerdo acerca de las convenciones a utilizar para la codificación. Suponiendo que se ha llegado a un acuerdo en cuanto al diseño, el programador de un módulo individual necesita saber:

1) La estructura y la organización de los archivos.
2) Las convenciones que se han acordado para la nomenclatura de las variables. A las variables más importantes, como, por ejemplo, las matrices que se utilicen a lo largo del programa, se les debe dar un nombre por anticipado. Se deberá acordar una convención para los nombres de las variables locales. A las variables que se pasen de un módulo a otro se les debe dar un nombre por anticipado o bien se debe diseñar una forma de asegurar que cada una sea única (agregándole el número del módulo original como sufijo, por ejemplo).
3) Cuáles son las rutinas de biblioteca que están a disposición del grupo, cuál es el formato de cada una de ellas, cómo se designan sus variables, qué hacen y en qué medida están comprobadas y depuradas.
4) Organización de las rutinas de tratamiento de errores (por ejemplo, si cada rutina se ocupa de sus propios errores o si las rutinas establecen un indicador de error, que es entonces tratado por la rutina de control).
5) La función exacta de cada módulo que se está escribiendo.
6) La gama y el tipo exactos de datos que cada módulo individual aceptará como entrada y devolverá como salida.

Esto supone un largo período de planificación con muchos encuentros para acordar la estrategia, seguido de un corto período de programación. De la comprobación nos ocuparemos más adelante.

MERGE

- El Spectrum posee la versión inmediata de la instrucción: fusiona el archivo especificado con el programa en memoria; en caso de coincidencia de números de línea, la línea entrante se escribe encima de la ya existente.
- Con la instrucción *SPOOL del BBC Micro se pueden crear versiones ASCII de los archivos de programas, luego escribir un programa en BASIC (o utilizar un procesador de textos) para acceder a estos archivos, a una línea de programa cada vez. Fusione los dos archivos en un tercer archivo ASCII y conviértalo en un programa utilizando la instrucción *EXEC.
- En el Commodore: OPEN 1,1:CMD1:LIST:PRINT #1:CLOSE 1 crea en cinta un archivo ASCII sin nombre del programa que esté cargado en memoria. Cargue (LOAD) el otro programa y añádale una rutina para entrar (INPUT) e imprimir las líneas del archivo ASCII en pantalla

Blanca elegancia

Esta vez examinamos el nuevo ordenador Apple IIc y analizamos la estrategia de marketing de la empresa

El éxito del Macintosh y la creciente competencia (por parte de Commodore en el mercado norteamericano del ordenador personal y de empresas como ACT e IBM en el mercado internacional de gestión) ha creado algunas dudas acerca del futuro de la gama de ordenadores Apple. Muchos distribuidores y analistas industriales vaticinaron que la gama se estaba aproximando al final de su vida en el mercado, a pesar de la insistencia de Apple en el sentido de que permanecería comprometida con la máquina 6502 y su gran base de usuarios. Para demostrar su apoyo, la empresa ha lanzado recientemente el Apple IIc, así como mejoras de software y hardware para las más antiguas líneas Apple II. Se espera que los nuevos productos prolonguen la vida del Apple II en el mercado en unos tres años.

En sus diversas formas (II, II+ y IIe), el Apple II ha ayudado a crear el mercado del ordenador personal, ha dominado las ventas de ordenadores en Estados Unidos durante varios años y ha contribuido a que las ventas totales de Apple superaran el billón de dólares. Existen más de dos millones de ordenadores Apple en uso en todo el mundo, a pesar de lo cual el Apple II jamás ha alcanzado el mismo nivel de éxito de ventas en el ámbito europeo, básicamente debido a una ineficaz política de precios y de comercialización. El precio de venta al público de la máquina era excesivamente elevado como para que se la considerara un ordenador personal. Y con frecuencia se alude a interferencias de las oficinas centrales de Apple en California como la razón por la cual Apple nunca ha conseguido el tipo de participación en los mercados británicos de gestión o educativo que ha obtenido en Estados Unidos. No obstante, el relativamente pequeño grupo de usuarios Apple de Gran Bretaña tiende a ser sumamente leal a la máquina.

Mejora portátil
La nueva versión del Apple II, mejorada y portátil, es el IIc. Tiene 128 K de RAM, una visualización de 80 columnas, una variedad de interfaces y una unidad de disco incorporada. En la fotografía lo vemos con la pantalla opcional de fósforo verde

Chris Stevens

La última reencarnación del Apple II es el IIc (la *c* alude a "compacto"). Es aproximadamente la mitad en tamaño que sus antecesores, a pesar de lo cual alberga una unidad de disco de 5 1/4 pulgadas a media altura a uno de los lados de su carcasa. Con su peso de 3,4 kg, el IIc es transportable y está claramente diseñado para utilizarlo durante el día en el trabajo y llevarlo luego a casa por la noche. Con este fin, el IIc tiene una pequeña asa para transporte moldeada en su carcasa plástica y una serie de conectores (para utilizar con una pantalla compuesta o RGB en el trabajo y un televisor normal en casa). El asa se acomoda en la carcasa para conformar un ángulo de trabajo cómodo. También hace que el aire circule alrededor de la máquina para evitar el sobrecalentamiento.

A diferencia de los modelos Apple II anteriores, el IIc es un sistema cerrado, sin ranuras para ampliación en su interior. En cambio, Apple ha incorporado en la máquina varias opciones de las más importantes. Éstas incluyen las puertas para visualización en pantalla y televisor; una puerta para palanca de mando que también soporta el ratón opcionalmente; una puerta para modem; una puerta para impresora; un conector para salida de audio, y un conector para una segunda unidad de disco. Las interfaces están etiquetadas con iconos (pequeñas representaciones gráficas de su función). El IIc también posee incorporada una visualización de 80 columnas y 128 Kbytes de RAM. La mayoría de estas características son opcionales en el IIe y exigirían la adición de al menos tres placas de ampliación.

El Apple IIc posee un teclado QWERTY de 63 teclas, con un trazado similar al del Apple IIe. La tecla Reset, sin embargo, se ha desplazado a una posición por encima del borde izquierdo del teclado y junto a ella se han colocado dos pequeños interruptores. El interruptor izquierdo conmuta la visualización en pantalla de 40 a 80 columnas. El manual del usuario recomienda utilizar una visualización de 40 columnas cuando se trabaje con un televisor, y una visualización de 80 columnas para una pantalla. (Parte del software Apple existente visualizará sólo 40, independientemente de la posición del interruptor.) El segundo interruptor conmuta

entre el juego de caracteres europeo y los caracteres norteamericanos que aparecen en el teclado. Esto es útil cuando se requiere un carácter que sólo se puede encontrar en uno de los dos juegos (como "#", que en el teclado europeo está reemplazado por "£"). Encima del borde derecho del teclado hay dos luces: una indica que el equipo está conectado a la red y la otra se ilumina cuando se está utilizando la unidad de disco.

Cuando se conecta el IIc, la unidad de disco comienza a girar automáticamente y espera un disco. Continuará girando hasta que se cargue un disco o hasta que se pulse la tecla Reset manteniendo deprimida la tecla Control. Sin ningún disco en la unidad, la IIc carga el BASIC Applesoft desde la ROM. El Applesoft permanece virtualmente sin modificaciones desde que se introdujera el Apple II+. Sólo tiene unas ligeras variaciones respecto a las primeras versiones del BASIC Microsoft estándar. La utilización del Applesoft permite que el IIc ejecute programas escritos para los modelos anteriores. Lamentablemente, el Applesoft carece de muchas de las estructuras de programación de que disponen las versiones más avanzadas, como el BASIC BBC. Por ejemplo, el Applesoft no posee la instrucción RANDOMIZE, facilidad para autonumeración de líneas, estructura IF...THEN...ELSE ni instrucción WHILE. También carece de las instrucciones CIRCLE y PAINT para la programación de gráficos.

Cuando en la unidad de disco hay un disco, el IIc ejecuta el DOS 3.3 (discos Apple II+y IIe) o bien el PRODOS, el nuevo sistema operativo de Apple. Éste es un derivado del sistema operativo que Apple diseñó para su primer sistema de gestión, el Apple III. El PRODOS posee una estructura de archivos jerárquica (árbol). Los archivos en disco se almacenan de forma muy parecida a los documentos que se guardan en un mueble archivador. Por consiguiente, los archivos relativos, por ejemplo, al proyecto ZED se podrían archivar en el disco bajo el encabezamiento ZED. Los archivos de contabilidad del proyecto ZED (como Costos, Ventas, Ganancias) se podrían reunir en un grupo denominado Cuentas. Con un sistema de archivo manual de este tipo, cuando se deseara hallar el archivo para Ventas, debería primero abrirse el archivo principal, ZED, después abrir el archivo Cuentas, y finalmente extraer el archivo etiquetado como Ventas. En el PRODOS esto se realiza a través de un "nombre de encaminamiento" *(pathname)* que lista los nombres de archivo apropiados por orden, de modo que el proceso que hemos descrito se cubriría entrando los nombres de los archivos, separados mediante barras:

```
/ZED/CUENTAS/VENTAS/
```

Los nombres de encaminamiento pueden ser de hasta 64 caracteres de longitud. Este proceso puede parecer complicado, y de hecho lleva un tiempo acostumbrarse a él, pero a la larga simplifica la organización y administración de los archivos en disco. Los sistemas de archivo de árbol como el PRODOS también se utilizan en el MS/DOS, el sistema operativo que emplea el IBM PC.

La visualización en pantalla del IIc también representa una mejora en relación a los modelos anteriores. Aparte de dos opciones de texto (24 líneas por 40 u 80 caracteres), el IIc posee tres pantallas para gráficos: 40 por 40 (baja resolución), 280 por

Un relato ejemplar

Chris Stevens

El Lisa

Apple III

Comprendiendo la importancia del mercado del ordenador de oficina, Apple desarrolló el Apple III, una brillante máquina de mesa que hace uso de dos procesadores 6502 para soportar hasta 512 K de memoria. El Apple III dispone de un sistema operativo llamado SOS (*Sophisticated Operating System:* sistema operativo sofisticado), que le permite comunicarse con una unidad de disco rígido y almacenar archivos en una estructura jerárquica. Esta estructura de árbol constituyó la base conceptual para el sistema operativo del Lisa, y muchas de sus características aparecieron en el MS/DOS del IBM PC. Lamentablemente, el III tuvo algunas dificultades de hardware inmediatamente después de su lanzamiento y eso le creó una mala reputación. Apple renovó y sustituyó todas las máquinas defectuosas, pero el III jamás superó la mala publicidad de que fue objeto ni la sensación general de que el sistema operativo era demasiado complicado. Como resultado, el Apple III desapareció del mercado.

El Lisa, introducido en 1983, fue el primer ordenador personal para el mercado masivo que utilizó software integrado con ventanas, tenía un sistema operativo basado en imágenes en vez de en palabras y se operaba mediante un dispositivo de control manual: el ratón. El Lisa suscitó tal expectación que hizo que la cotización en bolsa de la empresa casi se triplicara en unos pocos meses. Desarrollado a un costo de más de 50 millones de dólares, se le fijó un precio demasiado elevado y fue dirigido al mercado equivocado. Originalmente Apple dirigió el Lisa a los ejecutivos de las grandes corporaciones. Mientras algunos altos ejecutivos se han sentido muy impresionados por el Lisa, la mayoría de sus ventas se han ido a pequeñas compañías especializadas en publicidad, diseño gráfico y relaciones públicas. El Lisa no se vendió tan bien como se esperaba, y la cotización de las acciones de Apple descendió más allá de su valor anterior al lanzamiento del ordenador nuevamente en el lapso de unos pocos meses. Recientemente el Lisa ha sido reemplazado por el menos caro Lisa II. Bajo la guía de John Scully, venido de la multinacional Pepsi Cola para conducir Apple, la innovadora empresa parece haber aprendido de sus errores. El Macintosh se está vendiendo en enormes cantidades en todo el mundo, recuperando algunos de los costos de desarrollo del Lisa/Mac, y el IIc parece llamado a darle renovados bríos al principal producto de Apple, el Apple II.

192 (alta resolución) y 560 por 192, llamada Double Hi-Res (alta resolución doble). Hay 16 colores disponibles. Apple ofrece una pantalla de fósforo verde y se espera que introduzca pronto una visualización en cristal líquido (LCD). La pantalla LCD la fabricará Sharp para Apple y tendrá una visualización completa de 24 líneas por 80 caracteres. Se espera que pronto habrá disponible un paquete de pilas para el sistema, que permitirá que el Apple IIc se convierta en un ordenador totalmente portátil.

La mayor cualidad que tiene el IIc es su base de software. Para el Apple II se han escrito más de 17 000 programas. Si bien algunos de éstos sólo se venden en Norteamérica, uno todavía puede estar bastante seguro de que cualquier cosa que desee hacer con un Apple seguramente ya se habrá hecho y se habrá escrito el software adecuado. La base de software incluye algunos de los mejores programas de juegos del mundo (CHESS 7.0, ZORK, Simulador de vuelo Microsoft, Pinball Construction Set); una amplia variedad de programas de tratamiento de textos, hoja electrónica y base de datos; programas de contabilidad; programas para diseño de gráficos; programas científicos para control de laboratorio y programas educativos (desde libros de lectura para principiantes hasta cálculo avanzado).

Además del software II y IIe existente, Apple ha introducido un programa llamado Appleworks, un programa integrado de tratamiento de textos, hoja electrónica y base de datos con ventanas. Appleworks es bastante sofisticado y fácil de utilizar. El IIc viene con un disco de introducción al Appleworks, si bien no se trata de un ejemplar operativo del programa. Como es típico tratándose de Apple, la empresa da por sentado que finalmente usted deseará adquirir este paquete. Otros discos que se suministran con el sistema son: una introducción similar al LOGO Apple; Apple Presents Apple, introducción interactiva al sistema básico; una introducción muy simple a la programación en BASIC, y el disco de utilidades de sistemas PRODOS. También está disponible el MousePaint, un programa de dibujo activado por ratón basado en el MacPaint. El MousePaint se suministra con el ratón Apple II.

El IIc viene con un pequeño folleto que describe cómo instalar el sistema y una guía del usuario de 142 páginas que explica breve y claramente las operaciones del sistema y el empleo de los cinco discos que vienen con el ordenador. Los manuales están bien escritos e ilustrados a todo color. Es evidente que están pensados para el usuario novel.

El disco del IIc es muy atractivo y elegante. Apple abandonó el plástico beige utilizado para el II, el Macintosh y el Lisa en favor de un acabado blanco brillante. Las estrías que atraviesan la parte superior de la carcasa dejan que el aire fluya por los circuitos para mantener frío el sistema.

El Apple IIc, al igual que sus predecesores, es una gran máquina de sobremesa para la oficina. Con el agregado de la pantalla LCD y el paquete de pilas, cuya aparición se espera en breve, habrá de ganarse una reputación como compañero de trabajo útil y portátil. Si su precio hubiera sido más asequible, también podría haber sido un popular ordenador personal.

Disk Pack Apple
El Apple IIc viene con un paquete que contiene cinco discos. Cuatro de ellos son una introducción al funcionamiento de la máquina, la programación en BASIC y a programas de aplicaciones opcionales que Apple espera que uno compre. El quinto es el disco de operaciones del sistema con el PRODOS, el nuevo sistema de archivos en disco para la línea Apple II

Salida de video compuesto

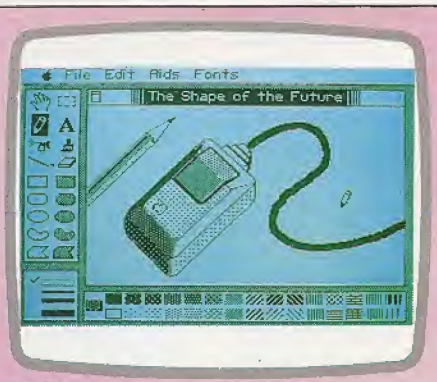
Puerta para impresora RS232

Fuente de alimentación
Es de 12 voltios pero aun así requiere un transformador para entrada de corriente de 240 voltios

Salida video RGB
Enchufando un pequeño adaptador PAL en esta puerta, se puede conectar el IIc a un aparato de televisión estándar

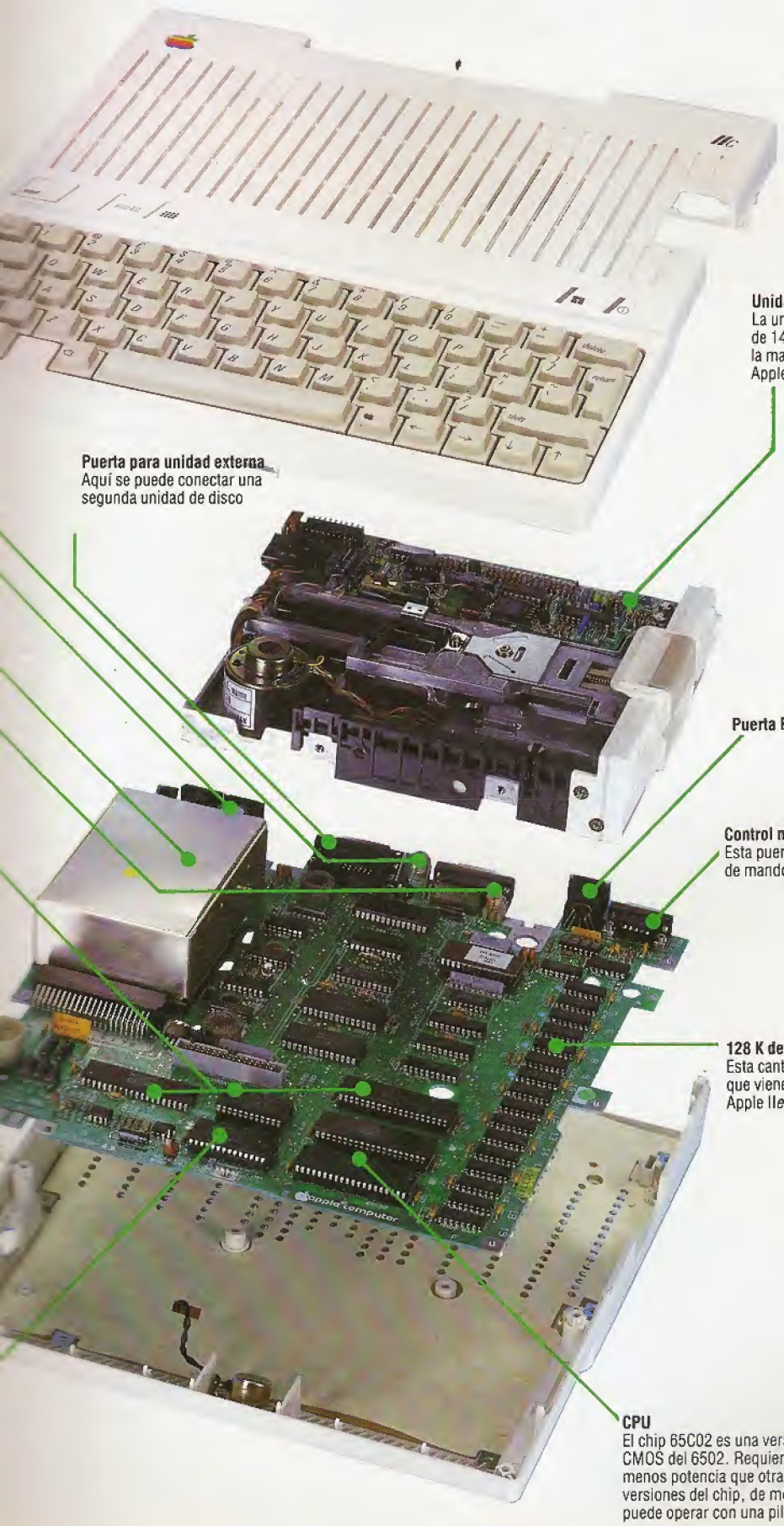
Controladores entrada-salida
Estos chips controlan las operaciones del teclado y las puertas de entrada y salida

Conector audio
Permite conectar al IIc con un amplificador hi-fi externo



MousePaint
El ratón Apple II está disponible para los modelos II+, IIe y IIc y viene con MousePaint. MousePaint está basado en el MacPaint, pero se trata de una versión a escala reducida del programa para el Macintosh. Permite que se utilice el ratón para crear imágenes muy fácilmente, y está escrito para sacar partido de la visualización en pantalla de alta resolución del IIc. Para utilizar el ratón con otros ordenadores Apple II, hay que adquirir una placa de interface extra que se enchufa en una de las ranuras para ampliación internas

ROM
Retiene el BASIC Applesoft y las rutinas del sistema



Unidad de disco
La unidad de disco incorporada de 143 K es compatible con la mayoría de los discos Apple II+ y IIe

Puerta para unidad externa
Aquí se puede conectar una segunda unidad de disco

Puerta RS232 para modem

Control manual
Esta puerta admite una palanca de mando o el ratón opcional

128 K de RAM para el usuario
Esta cantidad es el doble de la que viene como estándar con el Apple IIe

CPU
El chip 65C02 es una versión CMOS del 6502. Requiere menos potencia que otras versiones del chip, de modo que puede operar con una pila

APPLE II

DIMENSIONES
305×292×64 mm

CPU
6502 @ 1MHz

MEMORIA
128 K de RAM, 16 K de ROM

PANTALLA
24 líneas de 40 u 80 caracteres. Tres modalidades para visualización de gráficos con resolución máxima de 560×192 pixels y 16 colores

INTERFACES
Puerta de 9 patillas para palanca de mando que también funciona como puerta para ratón; puerta RS232 para modem; salida RGB (o PAL TV); salida para video compuesto; puerta para unidad de disco exterior, y puerta RS232 para impresora

LENGUAJES DISPONIBLES
BASIC Applesoft residente en ROM; LOGO, PASCAL, FORTRAN

TECLADO
Teclado de 63 teclas tipo máquina de escribir con cuatro teclas para cursor, juegos de caracteres internacionales

DOCUMENTACION
Con la máquina viene una guía del usuario a todo color y fácil de comprender. La guía está claramente diseñada para el usuario novel. Asimismo, hay dos delgados manuales para ayudar al usuario a instalar el IIc y trabajar con el disco de utilidades del sistema

VENTAJAS
La mayor cualidad del IIc la constituye su compatibilidad con el Apple II, lo que significa que puede ejecutar los más de 17 000 títulos disponibles para el otro Apple

DESVENTAJAS
El BASIC Applesoft, que en seis años no ha sufrido ninguna modificación sustancial y está empezando a hacer ostensible su edad, carece de flexibilidad. Puede que el precio del IIc lo coloque fuera del alcance de muchos usuarios personales

El mejor reparto

He aquí un programa que le permite crear su propio juego de caracteres para el Commodore 64

El Commodore 64 es capaz de producir espléndidos gráficos y sonido (tal como lo demuestra ampliamente gran parte del software comercial), pero su BASIC no dispone de ninguna instrucción de color ni de sonido incorporada con ese fin. Las instrucciones BEEP, DRAW, INK y PAPER de que dispone el BASIC del Spectrum, por ejemplo, carecen de equivalentes en el magro juego de instrucciones que tiene a su disposición el programador del Commodore 64. El resultado es que la mayoría de los programas en BASIC poseen sonido y gráficos muy elementales e incluso los mejores programas tienden a contener muchas sentencias DATA y POKE, tal como refleja el listado incluido en este capítulo. El programa generador de caracteres que listamos aquí hace que el proceso de definir nuevos caracteres sea menos penoso al permitir que el usuario los diseñe en la pantalla, en vez de colocar (POKE) valores directamente en la RAM; estas definiciones hechas en pantalla se colocan (POKE) luego automáticamente en la memoria.

Ya hemos investigado con cierto detalle el procedimiento que implica definir sus propios caracteres en el Commodore 64 (véase p. 712). Las acciones preliminares esenciales se realizan en la subrutina de la línea 61000. El tope de la memoria para el usuario se baja desde la posición 40959 a la 14335. El juego de caracteres en mayúsculas completo de dos Kbytes que el Commodore tiene residente en ROM (desde la dirección 53248 en adelante) se copia luego en RAM (desde 14336 en adelante), donde se puede acceder y manipular utilizando sentencias PEEK y POKE. Por último, se conecta el VIC (Video Interface Chip) para direccionar el juego de caracteres reubicado.

Una vez reubicado el juego de caracteres en RAM, la rutina de inicialización visualiza en la pantalla dos "ventanas" y el control pasa a la rutina de entrada de la línea 2500. Esta rutina explora el teclado y mantiene un cursor intermitente en la ventana izquierda o "de edición". En esta ventana se visualiza el carácter que se esté redefiniendo, con los valores de sus ocho bytes de definición junto al mismo.

Las teclas de función sin el uso de la tecla SHIFT (f1, f3, f5, f7) controlan el movimiento del cursor dentro de esta ventana. La celda de debajo del cursor (correspondiente a un bit de uno de los ocho bytes de definición) se puede activar o desactivar con la tecla de función f2 más la tecla SHIFT. Cuando esto sucede se actualizan los ochos valores de definición e inmediatamente se puede ver cómo cambian todas las ocurrencias del carácter en la pantalla.

La pulsación de la tecla de función f4 con SHIFT permite que se reemplace el carácter de la ventana de edición por otro carácter. Los caracteres se describen mediante sus valores POKE (o código de pantalla) según están listados en el manual del usuario. Estos valores no son los mismos que los códigos CHR$ (aunque existe una correspondencia), pero su utilización resulta más conveniente en este caso porque las definiciones de caracteres están dispuestas en la memoria por el orden de estos códigos.

La tecla de función f6 más SHIFT permite escribir una copia del carácter que se está editando en la ventana de la derecha (o "de texto"), en la posición correspondiente a la del cursor de edición y en su tamaño real. Si, por ejemplo, el cursor estuviera en el rincón superior izquierdo de la ventana de edición y se estuviera editando el carácter "A", entonces se escribiría "A" en el rincón superior izquierdo de la ventana de textos cuando se pulsara f6.

Por último, pulsando la tecla del signo de exclamación se interrumpe el programa; CONT lo reiniciará. Cuando se abandona el programa, se puede digitar NEW y después cargar (LOAD) otro programa sin que se altere su juego de caracteres redefinidos. No obstante, ello supone ciertos problemas. En primer lugar, el tope de la memoria para el usuario se ha rebajado, de modo que sólo hay 12 Kbytes disponibles para el nuevo programa. En segundo lugar, al apagar la máquina se destruye el nuevo juego. En un futuro capítulo analizaremos ambos problemas. Mientras tanto, apunte las definiciones de sus nuevos caracteres redefinidos y utilícelos, si fuera necesario, como se muestra en el programa de la página 713.

Sin adorno
En la ventana izquierda, o de edición, está la versión redefinida del carácter número 55: el carácter "7". Se le ha quitado el rabo de la izquierda y se le ha agregado la barra cruzada continental. En la ventana derecha, o de texto, hay cuatro copias del siete. Los cambios son visibles en estos sietes y en los de alrededor de las ventanas, pero el siete del recuadro de estado conserva su definición original

Pinzas
Los sprites son tan sólo múltiples caracteres definidos por el usuario. Se han redefinido nueve caracteres y colocado juntos en la ventana de textos. El símbolo de tanto por ciento es, en su forma redefinida, la sección central derecha del cangrejo

Ian McKinnell

Definiendo caracteres

```
 19 REM********** C64 **********
 20 REM*  GEN. DE CAR. DEF. POR EL USUARIO  *
 21 REM************************************
 50 POKE 52,56:POKE 56,56:CLR
 60 PRINT CHR$(147)"POR FAVOR ESPERE 22 SEG"
 70 GOSUB 61000: REM COPIAR JUEGO CAR.
100 GOSUB 1000: REM INICIALIZAR
120 FOR CT=0 TO 1 STEP 0
140 GOSUB 2500: REM INPUT
160 GOSUB 3000: REM VALIDAR
180 ON PT GOSUB 3500,4000,4500,7000
200 NEXT CT
900 END
999 REM************************************
1000 REM*          INICIALIZAR             *
1001 REM************************************
1020 DIM BD(8,8),C$(2,2),OF(2,7),CX(4)
1040 SH$=CHR$(19):SC$=CHR$(147):R$=CHR$(18):N$=
     CHR$(146):CD$=CHR$(17)
1060 P$=CD$+CD$+CD$:P$=P$+P$+P$+P$:P$=
     P$+P$:P$=SH$+P$
1080 C1$=CHR$(144):C2$=CHR$(5)
1200 REM  ------ INICIALIZAR PANTALLA --------
1210 S0=1024:PRINT SC$C1$
1220 R0=4:C0=3:RL=8:CL=8:OF=16
1230 Z$="CARACTERES DEFINIDOS POR EL USUARIO"
1240 C=4:GOSUB2100:R=24:GOSUB2100
1250 L$="  76543210 ":S$="      "
1260 Z$=L$:R=R0:C=C0:GOSUB 2100
1270 C=C0-1:FOR R=R0+1 TO R0+8
1280 Z$=STR$(R-R0-1):Z$=Z$+S$+Z$
1290 GOSUB2100:C=C+OF:GOSUB2100
1300 C=C-OF:NEXT R
1310 C=C0:Z$=L$:GOSUB2100
1320 L$="  01234567  "
1330 Z$=L$:R=R0:C=C0+OF:GOSUB 2100
1350 C=C0+OF:R=R0+CL+1  :GOSUB2100
1370 PRINT C2$
1400 C$(1,1)=R$+"  "+N$:C$(1,2)="  "
1410 C$(2,1)=R$+"**"+N$:C$(2,2)="**"
1420 REM  --------- DESPLAZ. CURSOR----------
1440 DATA 0,-1,+1,0,-1,0,0,+1
1460 FOR K=1 TO 2:FOR L=1 TO 4:
1480 READ OF(K,L):NEXT L,K
1500 REM  ------ CONVERSION CARACS. --------
1520 DATA 64,0,32,64
1540 FOR K=0 TO 3:READ CX(K):NEXT K
1620 RP=1:CP=1:CN=1:GOSUB 6000
1990 RETURN
1999 REM************************************
2000 REM*          PONER CRSR @ R, C       *
2001 REM************************************
2050 PRINT LEFT$(P$,R+1)TAB(C);
2070 RETURN
2099 REM************************************
2100 REM*          PRINT Z$ @ R,C          *
2101 REM************************************
2150 PRINT LEFT$(P$,R+1)TAB(C)Z$;
2170 RETURN
2499 REM************************************
2500 REM*       INTERMIT. CRSR @ RP,CP     *
2501 REM************************************
2520 CF=1+BD(RP,CP):R=RP+R0:C=CP+C0
2540 FOR LP=0 TO 1 STEP 0
2560 FOR CS=1 TO 2:DE=10:GOSUB 2800
2580 GET GT$
2600 IF GT$ <> "" THEN LP=1:CS=2
2620 Z$=C$(CF,CS):GOSUB 2100
2640 DE=10: GOSUB 2800
2660 NEXT CS,LP:RETURN
2799 REM************************************
2800 REM*          DEMORA PARA DE          *
2801 REM************************************
2820 FOR NN=1 TO DE:NEXT:RETURN
2999 REM************************************
3000 REM*          VALIDAR ENTRADA         *
3001 REM************************************
3020 IF GT$="!" THEN R=18:C=0:GOSUB2000:STOP
3040 GT=ASC(GT$)-132:F=2*INT(GT/2)
3060 IF (GT<1)OR(GT>8) THEN PT=0:RETURN
3080 IF GT<5 THEN PT=1:RETURN
3100 PT=GT-3
3490 RETURN
3499 REM************************************
3500 REM*          MOVER EL CURSOR         *
3501 REM************************************
3520 NY=RP+OF(2,GT):NX=CP+OF(1,GT)
3540 IF (NY<1)OR(NY>RL) THEN RETURN
3560 IF (NX<1)OR(NX>CL) THEN RETURN
3580 RP=NY:CP=NX
3620 RETURN
3999 REM************************************
4000 REM*          ACTIVAR UNA CELDA       *
4001 REM************************************
4020 TG=1-BD(RP,CP):Z$=C$(1+TG,2)
4040 R=R0+RP:C=C0+CP:GOSUB2100
4060 BD(RP,CP)=TG:MP=NCGEN+CN*8-1
4120 PE=PEEK(MP+RP)
4140 PE=PE+(TG*2-1)*(2^(8-CP))
4160 POKE (MP+RP),PE:GOSUB6500:RETURN
4499 REM************************************
4500 REM*       DEFINIR NUEVO CARACTER     *
4501 REM************************************
4520 FOR K=1 TO 1
4540 Z$=R$+"      CAMBIAR CARAC.      "
4550 R=14:C=7:GOSUB2100
4560 Z$="   NUEVO NUMERO   "
4570 R=15:GOSUB2100
4580 C=19:GOSUB2000:INPUT A$
4600 CN=VAL(A$):PRINT C2$
4620 IF (CN<0)OR(CN>127) THEN K=0
4640 NEXT K:GOSUB 6000
4660 Z$="          ":C=7
4670 R=14:GOSUB2100:R=15:GOSUB2100
4680 RETURN
5999 REM************************************
6000 REM*          VISUALIZAR CARAC.       *
6001 REM************************************
6020 MP=NCGEN+CN*8-1
6040 FOR RP=1 TO 8:PE=PEEK(MP+RP):Z$=""
6060 FOR CP=8 TO 1 STEP -1:N=INT(PE/2)
6080 Q=PE-2*N:BD(RP,CP)=Q:PE=N
6100 Z$=C$(Q+1,2)+Z$:NEXT CP
6120 R=R0+RP:C=C0+1:GOSUB2100
6130 NEXT RP
6140 X$=CHR$(CN+CX(INT(CN/32)))
6150 Z$=R$+"CARACTER":R=1:C=11
6160 GOSUB2100:R=R+2:GOSUB2100
6170 Z$="  "+X$+"=   "
6180 R=R-1:GOSUB2100
6190 C=C+4:Z$=STR$(CN)+N$:GOSUB2100
6220 RP=1:CP=1
6490 GOSUB6500:RETURN
6499 REM************************************
6500 REM*          VISUALIZAR BYTES        *
6501 REM************************************
6540 C=C0+CL+2:FOR R=R0+1 TO R0+8
6560 Z$=STR$(PEEK(MP+R-R0))+"  "
6580 GOSUB2100:NEXT:RETURN
6999 REM************************************
7000 REM*          COLOCAR UN CARAC.       *
7001 REM************************************
7020 Z$=X$:C=C+OF:GOSUB2100:RETURN
60999 REM***********************************
61000 REM*       REUBICAR CARAC. GEN.       *
61001 REM***********************************
61100 CGEN=53248:NCGEN=14336
61120 ITRPT=56334:IOPT=1:PO=53272
61125 RETURN
61150 POKE IT,PEEK(IT)AND254
61200 POKE IO,PEEK(IO)AND251
61250 FOR J=0 TO 2047
61300 POKE (NCGEN+J),PEEK(CGEN+J)
61350 NEXT J
61400 POKEIO,PEEK(IO)0R4
61450 POKE IT,PEEK(IT)OR1
61500 POKE PO,(PEEK(PO)AND240)OR14
61990 RETURN
```

Encender y apagar

Continuamos explicándole cómo construir un dispositivo interface que permita a su ordenador controlar pequeños aparatos caseros

Ian McKinnell

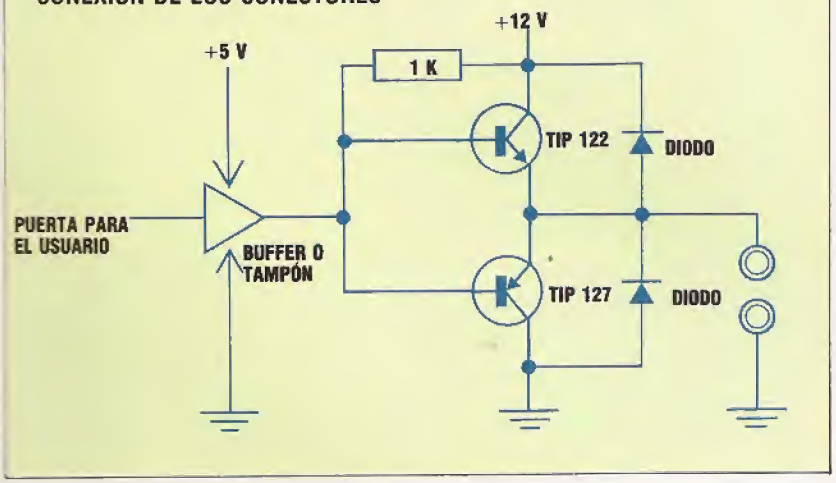
La caja
Ésta es la caja terminada, mostrando el enchufe minicon y los cables de entrada-salida. Se debe tener cuidado al cortar la placa y la ranura de la caja de modo que la placa no se mueva en la caja cuando se inserta el enchufe en el conector

Esta interface posibilitará que su ordenador personal controle dispositivos de poco voltaje que consumen poca corriente. La salida de cada uno de los cuatro bits de la puerta para el usuario del ordenador es tamponada por el mismo chip de buffer que utilizamos en la construcción de una caja buffer en un capítulo anterior de este apartado (véase p. 1003). La salida del mismo se utiliza para conmutar los transistores, que son capaces de controlar voltajes y potencias superiores de los que puede manipular el buffer. Los dos diodos de la salida protegen a los transistores de corrientes inversas que pueden crear algunas cargas inductoras, como relés y motores.

La interface se puede utilizar como controlador de motores bidireccionales. Para conectar o desconectar uno de esos motores basta con colocarlo entre el conector de salida y el conector a tierra. Cuando sale un uno desde el ordenador, el motor arranca. Una salida cero apagará el motor.

La conexión del motor entre dos de las salidas de la interface, sin embargo, permitirá controlar también la dirección del movimiento del motor. Si el ordenador envía las mismas salidas (ambas ceros o ambas unos), entonces en ambas salidas de la interface aparecerá el mismo voltaje y a través del motor no fluirá ninguna corriente. Un uno en una salida y un cero en la otra darán una diferencia en voltajes que hará que el motor gire en una dirección. Invirtiendo la lógica, la corriente fluirá (y el motor girará) en la dirección contraria.

CONEXIÓN DE LOS CONECTORES



Liz Dixon

Construcción de la interface

En primer lugar, construya la caja en la cual se alojará la interface. La placa de circuitos encajará exactamente en la caja especificada para el proyecto de la caja buffer. La caja debe estar perforada para aceptar los conectores y el enchufe del bus (y el conector minicon si así se requiriera).

Después de haber fijado los conectores en la caja tenemos que hacer las conexiones entre ellos. Utilizando un trozo de cable estañado, conecte todos los conectores a tierra (negros) juntos; tome un trozo de 15 cm de cable plano de nueve vías y fije una hebra al cable estañado que conecta los conectores a tierra. Fije las otras ocho hebras, de a dos, a cada uno de los cuatro conectores de salida (rojos).

Ahora corte la veroboard a la medida correcta (45 agujeros por 16 franjas) y quite la media fila de agujeros de uno de los extremos. Conserve este retal de la placa, porque se utilizará para la construcción de la otra interface. Ahora efectúe los cortes de pistas tal como se ilustra en la fotografía A.

Suelde primero los componentes pasivos: el conector del chip, el conector del bus y los enlaces de cables. Si desea instalar el conector de ampliación del bus, instálelo ahora, pero deje para el final el cable que conecta al mismo con el enchufe del bus.

A continuación suelde las resistencias, seguidas de los diodos. Compruebe y asegúrese de que los mismos quedan instalados en la posición correcta. Seguidamente suelde los ocho transistores. Por último, suelde las conexiones a los conectores tal como se indica en la fotografía B e instale el chip en su lugar. Ahora puede ensamblar la placa en la caja y la interface ya está lista para utilizar.

Lista de componentes

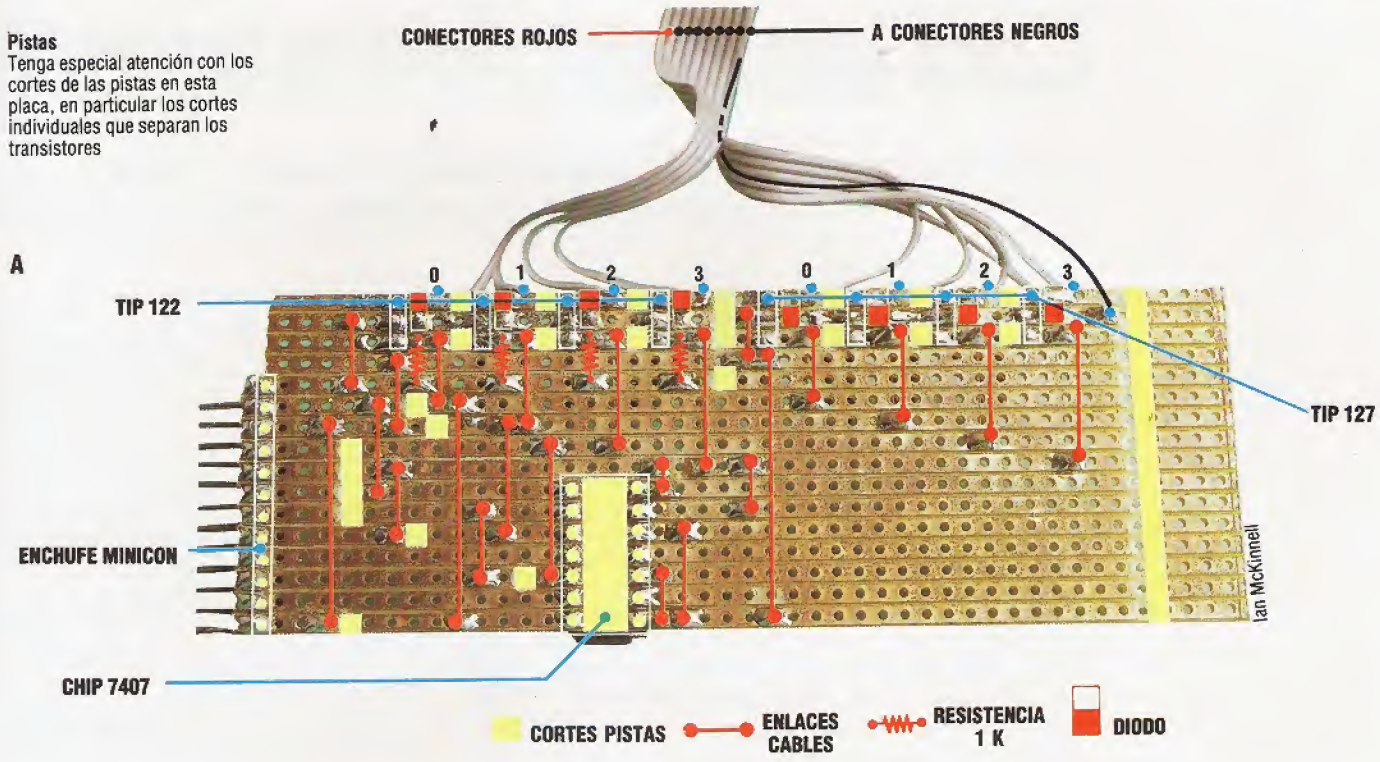
Cantidad	Artículo
1	Veroboard 50 agujeros×24 líneas
4	Resistencia 1 K-ohm
8	Diodo 1N4001
4	Transistor TIP 122
4	Transistor TIP 127
1	7407
1	Conector DIL 14 patillas
4	Conector 4 mm rojo
4	Conector 4 mm negro
4	Enchufe 4 mm rojo
4	Enchufe 4 mm negro
1	Enchufe minicon 12 vías
1	Caja "2004" 120×65×40 mm
1	Conector minicon 12 vías*

* El último artículo es opcional. Amplía el bus del sistema más allá de esta interface, de modo que se puedan enchufar simultáneamente otras.

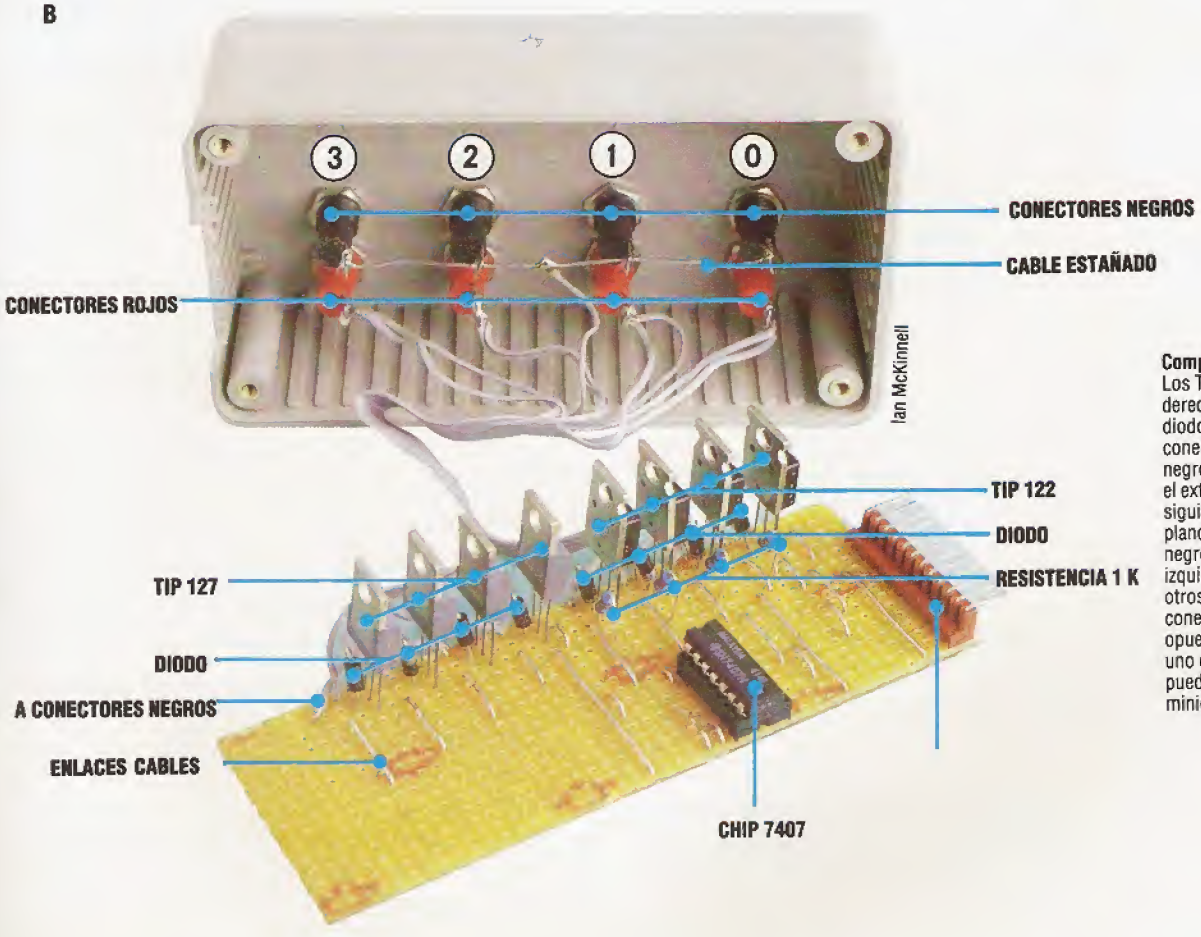
Debe contar con un poco de cable estañado y de cable plano que posiblemente le hayan sobrado del proyecto anterior.

Pistas
Tenga especial atención con los cortes de las pistas en esta placa, en particular los cortes individuales que separan los transistores

A



B



Componentes
Los TIP 122 están en el extremo derecho de la placa y los cuatro diodos entre ellos están conectados con el extremo negro a la pista del borde y con el extremo plateado a la pista siguiente. La línea de cable plano desde los conectores negros está soldada al extremo izquierdo de esta pista. Los otros cuatro diodos están conectados en el sentido opuesto entre los TIP 127. A uno de los lados de la placa se puede ver claramente el enchufe minicon

Los bucles, otra vez

Analicemos otra forma de establecer iteraciones en el código, similar en muchos aspectos al uso de las bifurcaciones

En anteriores capítulos, hemos comprobado cómo puede efectuarse una misma parte de un programa. Esto se consigue con el uso de las bifurcaciones, ya sean condicionales o incondicionales. Pues bien, existe una forma similar de establecer iteraciones en el código mediante las técnicas reconocidas por el BASIC como FOR...NEXT (general para todas las versiones) y WHILE...WEND (propia del tipo Microsoft y sus derivados), basadas en los mismos principios pero con claras diferencias. Establezcamos los puntos comunes entre un bucle tratado con FOR...NEXT y otro en la forma tradicional. El ejemplo en ambos casos será el mismo: la impresión de los números pares comprendidos entre 2 y 50. La figura 1 muestra el ordinograma junto con la transcripción del mismo a BASIC, tal como solemos hacerlo. Los componentes, tanto en una como en otra solución, son similares: un valor inicial (2) de la variable, y otro valor (50), final, que marca el tope que debe alcanzar dicha variable y, en este caso, asimismo el límite superior a visualizar. Por último, un valor de incremento (2), factor que va incrementando la variable hasta alcanzar el valor final.

En la línea 50 de la figura 1 se establece la comparación de la variable con el valor final para adoptar una decisión ante sus dos posibles salidas: final de proceso o continuación-iteración.

En la figura 2 puede observarse la existencia de los mencionados elementos, salvo el de la decisión (IF), ya que el ciclo FOR...NEXT lleva un contador interno que marca cuándo debe repetir el bucle o bien seguir en secuencia a la próxima instrucción. Así, y a modo de adaptación libre, se podría comentar el programa de la figura 2 diciendo: escribir repetidamente el contenido de la variable A empezando con un valor inicial 2, en incrementos de 2 unidades hasta alcanzar el valor 50.

La representación del FOR...NEXT, WHILE...WEND en diagramación utiliza dos símbolos complementarios:

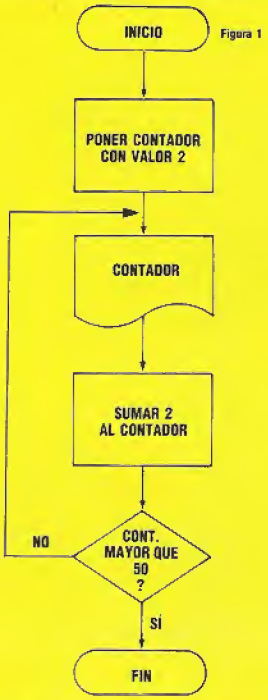
para apertura, frase inicio (FOR-WHILE)

y

para cierre de ciclo (NEXT-WEND)

También es posible (en caso de FOR) la especificación del incremento (STEP) de la variable de control del bucle.

Figura 1




```
10 REM FORMA HABITUAL DE EJECUCION DE BUCLES
20 A=2
30 PRINT A
40 A=A+2
50 IF A > 50 THEN END
60 GOTO 30
```

Figura 2


```
10 REM BUCLE MEDIANTE FOR...NEXT
20 FOR A=2 TO 50 STEP 2
30 PRINT A
40 NEXT A
50 END
```

Simple aritmética

Ya podemos examinar algunos programas con instrucciones del 6809 que realizan operaciones sencillas de aritmética

A este nivel del curso va siendo hora de juntar varias instrucciones en un solo programa que sea operativo, aunque para ello necesitamos algunas nuevas que todavía no conocemos junto con las maneras de representar datos. Comenzaremos por delinear un programa sencillo que sirva para convertir un número escrito en BCD (*Binary Coded Decimal*: decimal codificado en binario) en su equivalente binario.

Un número decimal codificado en binario es una manera muy útil de representar los números decimales en forma binaria cuando trabajamos con procesadores de ocho bits. Mediante esta representación, cada dígito en decimal se convierte en su equivalente binario. Por ejemplo, el número decimal 69 equivale en BCD a %01101001: los cuatro primeros bits, empezando por la derecha (1001), representan el 9 en binario, como sabe el lector, y los cuatro restantes (0110) son el 6. Nótese que si tomáramos %01101001 como un todo en binario su equivalente decimal no es 69 sino 105.

Nuestro programa convertidor empleará, entre otras, las siguientes instrucciones:

• LSR (*Logical Shift Right*: desplazamiento lógico a la derecha): Desplaza cada uno de los bits del operando un lugar a la derecha. El bit que ocupa el lugar más extremo a la derecha se separa y se coloca en el registro de código de condición del procesador como bit de arrastre. El lugar "vacío" que obviamente queda en el extremo izquierdo del operando se rellena con un cero.

• AND: Realiza la operación lógica AND sobre cada bit de un registro con los bits correspondientes del operando, quedando el resultado en el registro. Es una instrucción muy utilizada con el fin de "enmascarar" determinados bits: si un registro posee un bit puesto en 1, aplicándole AND con otro bit prevalecerá este último bit en el registro (recuérdese, 1 AND 0=0 y 1 AND 1=1). En el caso de que el bit del registro contenga un 0, el resultado será siempre 0. Por ejemplo si operamos un valor como %00001111 contenido en algún registro con el valor correspondiente a alguna posición de memoria mediante la operación AND de seguro obtendremos al menos para los cuatro bits más a la izquierda esos mismos ceros. Así:

```
%00001111   valor del registro
%10110110   valor de la posición de memoria AND
%00000110   resultado que queda en el registro
```

• MUL: Multiplica los contenidos de los registros A y B, llevando el resultado al registro D (que, como se dijo, está formado por la unión de los registros A y B y, por tanto, es de 16 bits). Hay muy pocos procesadores de ocho bits capaces de aceptar la multiplicación como un opcode.

• SWI (*SoftWare Interrupt*: interrupción por software): Es el modo más aconsejable de concluir un programa en código máquina y devolver el control al sistema operativo. Examinaremos con mayor detalle esta instrucción más adelante.

He aquí el programa "De BCD a binario":

1) Especificación de un valor para el contador de posiciones:

```
ORG $1000
```

2) Almacenamiento del número 58 en BCD dentro de la posición BCDNUM al tiempo que se reserva un byte en BINNUM:

```
BCDNUM  FCB   %01011000
BINNUM  RMB   1
```

3) Carga del número 58 en BCD dentro del registro A enmascarando el dígito inferior, y almacenamiento de éste en BINNUM:

```
START LDA   BCDNUM
      ANDA  #%00001111
      STA   BINNUM
```

4) Carga del 58 en BCD en el acumulador A y desplazamiento del dígito superior (o sea, los cuatro bits más a la izquierda) hacia la derecha:

```
SHIFT LSRA
      LSRA
      LSRA
      LSRA
```

5) Carga del número 10 (en decimal) en el registro B y multiplicación de éste por el contenido de A:

```
MULT  LDB #10
      MUL
```

6) El resultado será de 16 bits y queda en el registro D, pero como el resultado no puede ser mayor que 90 (10×9=90), sólo se necesita el byte inferior de D. Este byte se encuentra en el registro B, por eso el paso siguiente será sumar el contenido de B a BINNUM y almacenar el resultado:

```
ADDIT  ADDB BINNUM
       STB  BINNUM
```

7) Tenemos, pues, el número en BCD almacenado en BCDNUM y su equivalente binario almacenado en BINNUM. Devolveremos (*return*) el control al sistema operativo y finalizaremos (*end*) el programa:

```
RETURN  SWI
END
```

El complemento a dos

Hasta ahora hemos descrito programas que sólo realizaban sencillas operaciones aritméticas, y por este camino continuaremos durante algún trecho

Decimal	Binario
−7	1001
−6	1010
−5	1011
−4	1100
−3	1101
−2	1110
−1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

más. Nos interesa ahora estudiar el problema del *signo*, o sea los números negativos y positivos.

El método más conocido de representación de los números negativos dentro de una posición de memoria o de un registro es denominado *complemento a dos*. Es fácil obtener dicho complemento: se invierten todos los dígitos (los ceros se cambian en unos, y viceversa) y después se le suma un uno al número así obtenido. Por ejemplo, el complemento a dos de 0101 se obtiene así: 1010+1=1011.

Pero ¿cuál es su utilización en las operaciones matemáticas con números negativos? Ante todo, consideremos cuántos números se pueden representar: un registro de ocho bits sólo acepta 256 diferentes combinaciones de bits (2^8), combinaciones que podríamos distribuir así: los 127 primeros números serán positivos, los 128 restantes negativos, o sea, desde el −128 hasta el +127 (nótese que con un registro de 16 bits iríamos desde el −32768 hasta el +32767), pues admite 2^{16}=65536 números distintos). En la tabla adjunta mostramos la representación de valores desde el −7 hasta el +7, empleando sólo cuatro bits.

Si observa usted dicha tabla, notará que todos los números negativos tienen el MSB (*Most Significant Bit*: el bit más significativo, o sea, el situado más a la izquierda) puesto en 1. Igualmente para los positivos el MSB está puesto a 0.

Podemos entonces definir algunas propiedades básicas relacionadas con el complemento a dos y la aritmética de los signos:

- El complemento a dos de un número negativo nos da su positivo, y al revés.
- El bit más significativo (MSB) está a 0 para los números positivos, y a 1 para los negativos. Ésta es la contraseña por la que reconoceremos fácilmente si un número es positivo o negativo.
- El complemento a dos del número cero es cero (sume usted la unidad a 1111, p. ej.).
- La suma y la resta pueden realizarse del mismo modo, teniendo el resultado el signo correcto.

Puede que a usted le tiente probar esta última propiedad en algunos ejemplos de sumas y restas. Inténtelo. Desgraciadamente la multiplicación resulta algo más difícil cuando se usan números negativos. La instrucción MUL que utilizamos en el programa anterior para convertir BCD en binario suponía que los registros A y B contenían valores sin signo. Si deseamos multiplicar dos números con sus respectivos signos + o − (o sea, 0 o 1 en el MSB) debemos recurrir a un programa.

Cualquier lector con alguna práctica de programación sabe lo limitados que resultan los programas "lineales" hasta aquí empleados. Sólo podemos conseguir algo más útil si nos servimos de alguna de las formas básicas de estructura de control:

- Selección: en la que escogemos entre dos alternativas de acción (semejante al IF del BASIC).
- Repetición: por la que repetimos una secuencia de operaciones:

1) mientras (*while*) se cumplan ciertas condiciones (la estructura WHILE...WEND)
2) hasta que (*until*) se cumplan ciertas condiciones (la estructura REPEAT...UNTIL); o bien
3) un cierto número de veces (FOR...NEXT).

Todas estas estructuras dependen de la habilidad para verificar si una condición es verdadera o falsa, siendo la condición más común el que una variable tenga o no un determinado valor. En assembly necesitamos utilizar estas estructuras, y deberemos ser capaces de verificar los valores de los registros. Por lo general se pueden verificar directamente según dos posibilidades (si un valor es cero o no, si es positivo o negativo). Con otras instrucciones adicionales podemos, no obstante, realizar otros tipos de verificaciones.

El registro de código de condición

Lo anterior se consigue con el empleo de este registro de código de condición (CC), que ya se mencionó de pasada anteriormente (véase p. 1018). Se trata de un registro de ocho bits, pero de él, al contrario de los restantes registros, no nos interesa el valor que almacena. Nos interesa más bien el estado (1 o 0) de cada uno de sus ocho bits. Cinco de los ocho bits sirven para expresar las condiciones que hemos venido analizando hasta aquí, los otros tres se encargan del tratamiento de las interrupciones (que examinaremos más adelante). Uno de los cinco, el H (el flag de arrastre *mitad*; en inglés, *Half*), casi es exclusivo de las operaciones aritméticas en BCD, y por el momento no nos va a preocupar. Los cuatro restantes, que sí nos importan a este nivel, son:

- C: Flag de arrastre (inglés: *Carry*) que sirve para retener el bit de arrastre (o de sustracción, en caso de la resta) tomado del bit más significativo tras una operación aritmética. Tiene también una función muy útil en caso de que queramos desplazar el contenido de un acumulador un bit nada más; varias de las operaciones de desplazamiento colocan el bit que se desecha en C. Este bit puede servir, por ejemplo, para verificar si el número es par o impar simplemente llevando el bit menos significativo a C y comprobando su valor. Se trata del bit 0 (el bit menos significativo) en el CC.
- V: Flag de desbordamiento (inglés: *oVerflow*), puesto a 1 cuando el resultado de una operación aritmética no cabe, por excesivamente grande, dentro del registro que debería contenerlo. Es el bit 1 en el CC.
- Z: Es el flag cero (inglés: *Zero*), puesto a 1 cuando el contenido de un registro es cero. Se trata del bit 2 en el CC.
- N: Flag de los negativos. Es una copia del bit más significativo (el bit del signo) del contenido de un registro; o sea, se pone a 1 si el número es negativo. Se trata del bit 3 en el CC.

Uno de los aspectos más difíciles del lenguaje assembly a la hora de programar, es atender el estado de los flags. No todas las instrucciones activarán los flags, y algunos de éstos se activan según el contenido del acumulador mientras otros pueden también sufrir cambios según otros registros. El procedimiento más seguro es verificar sólo los valores contenidos en un acumulador, y realizarlo en el momento en que aparece el valor requerido, ya que es difícil asegurar que los flags no cambien con la intervención de cualquier otra instrucción.

Los flags son verificados por medio de instrucciones de bifurcación, las equivalentes a bajo nivel, del GOTO en BASIC. El 6809 emplea bifurcaciones relativas (más que absolutas) de modo casi exclusivo. La diferencia está en que una bifurcación relativa transfiere el control un cierto número de bytes hacia adelante (o hacia atrás), mientras que una bifurcación absoluta transfiere el control a una dirección absoluta especificada. Pero el efecto es el mismo. Hay que distinguir entre bifurcaciones *cortas*, cuando la amplitud se puede expresar en un solo byte (del −128 al 127), y bifurcaciones *largas*, que pueden dirigir a cualquier punto de la memoria. Continuaremos usando tan sólo bifurcaciones cortas.

El 6809 tiene un extenso conjunto de instrucciones de bifurcación, que iremos presentando a medida que las vayamos necesitando. Los ejemplos que siguen ilustran las instrucciones empleadas para verificar y comparar los valores retenidos en los acumuladores. Ilustran igualmente el uso de las instrucciones de bifurcación en los procedimientos de selección y repetición.

• ANDCC: No es posible cargar valores directamente dentro del registro de código de condición, pero es una buena práctica poner a cero todos los flags que va usted a necesitar antes de comenzar a usarlos. La forma más fácil de hacer esto es servirse de la instrucción ANDCC, que opera como la orden AND, empleando ceros como máscaras en las posiciones de bit que deseamos usar.

• SUB (SUB*tract*: resta): Resta el operando del contenido del acumulador, y afecta a los flags C, V, Z y N, según el resultado de la operación (incluso el flag H se activa si la resta es de ocho bits).

• CMP (CoMParación): Funciona igual que SUB, sólo que no cambia el contenido del registro. Afecta, como en SUB, a los flags C, V, Z, N (y H).

• BRA (*Inconditional* BRA*nch*: bifurcación incondicional): Parecida al GOTO del BASIC.

• BGT (B*ranch if* G*reater* T*han zero*: bifurcar si mayor que cero): Se comprueba con ella el signo de los números. La bifurcación tiene lugar si Z es cero (el número no es cero). En previsión de que el signo pueda ser incorrectamente interpretado si hubo desbordamiento, N y Z deben ser cero (directamente, no negativos) o bien N y V deben ser uno (negativo equivocado por causa del desbordamiento). Otras verificaciones similares para los números con signo son BGE, BLT y BLE ("bifurcar si mayor o igual", "si menor que" y "si menor o igual").

• BLO (B*ranch if* LO*wer than zero*: bifurcar si es menor que cero): Se trata de una verificación sin signo, pues no tiene sentido comprobar N para un número sin signo. La bifurcación se produce cuando el flag C se pone a 1, el cual indica la unidad a quitar en una resta. Verificaciones similares son BLS, BHI y BHS.

• He aquí un programa para encontrar el número más grande entre dos con signo y de ocho bits almacenados en $3000 y $3001. El resultado se colocará en $3002:

1) Etiquetaremos los números:

```
NUM1    EQU     $3000
NUM2    EQU     $3001
RES     EQU     $3002
        ORG     $1000
```

2) Iniciamos la codificación: debemos poner a cero todos los flags del código de condición y cargar el primer número. Éste se compara con el otro:

```
        ANDCC   %11110000
        LDA     NUM1
        CMPA    NUM2
```

3) Si el más grande es NUM1, el programa bifurcará a FIN, de lo contrario cargará el segundo número en el registro A. Cualquiera que sea el número que se halle en el registro cuando se pase a FIN, irá a almacenarse a RES, y el programa devolverá el control al sistema operativo, concluyendo (END), entonces, de la siguiente manera:

```
        BGT     FIN
        LDA     NUM2
FIN     STA     RES
        SWI
        END
```

Directivas originales
Los diferentes efectos que las directivas del ensamblador y las sentencias en assembly tienen sobre el contador de posiciones y sobre el contenido de la memoria pueden apreciarse en el siguiente ejemplo

Directivas originales

```
CAMPO DE ETIQUETA  CAMPO OPCODE  CAMPO OPERANDOS   CONTADOR POSICIONES  CONTENIDO MEMORIA
*----------DEMONSTRATION----------
RESET     EQU     $F100          $0400                ??
INDEX     EQU     16             $0400                ??
MASK1     EQU     %01101010      $0400                ??

          ORG     $1000          $1000                ??

CR        FCB     16             $1000                $10

MEMTOP    FDB     $7FFF          $1001                $7F
                                 $1002                $FF

TABLE1    RMB     7              $1003                ??
                                 $1004                ??
                                 $1005                ??
                                 $1006                ??
                                 $1007                ??
                                 $1008                ??
                                 $1009                ??

ERRMSG    FCC     'ERROR         $100A                $45
                                 $100B                $52
                                 $100C                $52
                                 $100D                $4F
                                 $100E                $52

          CLRA                   $100F                $4F

          END                    $100F                ??

*-------SYMBOL TABLE----------
RESET   F100   INDEX   0010   MASK1   006A
CR      1000   MEMTOP  1001   TABLE1  1003
ERRMSG  100A
```

Como no se puso ningún ORG, la dirección de la posición es la que el ensamblador tiene por defecto. La dirección no queda afectada por EQU, y el contenido de la memoria está todavía por definir

Sitúa la posición según se indica, pero el contenido de la memoria sigue indefinido

FCB hace que el operando quede almacenado en el byte direccionado por el contador de posiciones

El FDB inicializa dos bytes

RMB reserva 7 bytes (de contenido indefinido) incrementando el contador de posiciones en ese número

Los códigos ASCII del operando string son colocados en la memoria gracias a la directiva FCC

¡Por fin una operación en assembly! No hay operando alguno; disponemos sólo de un byte como opcode

Otra directiva que no afecta al contador de posiciones

Así es como se almacenarán los símbolos utilizados en el programa dentro del espacio de trabajo del ensamblador para uso propio durante el assembly

Una nueva meta

Memotech, proveedor de accesorios y periféricos para el Sinclair ZX81, ya está produciendo su propia gama de micros: la serie MTX

Línea especial
Memotech empezó con productos especializados para el Sinclair ZX81, como el Memopack, que vemos en la fotografía, que le proporcionaba al ZX81 32 K adicionales de Ram

Memotech se creó a raíz del enorme interés del público por los primeros microordenadores de Sinclair Research. A pesar de la popularidad del ZX80 y del ZX81, pronto se hizo evidente que las máquinas estaban severamente limitadas por una carencia de memoria, y se creó un inmenso mercado para placas de memoria adicionales.

Los dos fundadores de esta empresa británica eran catedráticos de la Universidad de Oxford: Geoff Boyd disertaba sobre metalurgia en Wilson College, y Robert Branton enseñaba matemáticas en Christ Church. Los dos se encontraron por primera vez en una muestra de informática en la universidad, en 1981, y decidieron trabajar juntos en accesorios para el ZX81. Su primer producto fue una placa de ampliación de 16 Kbytes, que fue seguido por una serie completa de "Memopacks", incluyendo paquetes de RAM de 32 Kbytes y 64 Kbytes, el paquete para gráficos en alta resolución (HRG), un análisis de hoja electrónica (Memocalc), un procesador de textos (Memotext), interfaces Centronics y RS232 y un teclado.

La gama MTX se lanzó oficialmente en febrero de 1984 y la empresa afirma que desde entonces se han vendido alrededor de 25 000 máquinas. Al igual que el BBC Micro, el MTX viene en dos modelos: el MTX500, de 32 Kbytes, y el MTX512, de 64 Kbytes. Las máquinas utilizan un microprocesador Z80A y ofrecen 16 colores en modalidad de alta resolución (256×192 pixels). El BASIC MTX es similar al BASIC BBC. El ordenador también dispone de un ensamblador-desensamblador.

El ordenador también se puede ampliar para hacer uso del paquete para gráficos HRX de Memotech. Partiendo de un MTX500 sin ampliar, el usuario puede añadir unidades de disco y las tres placas controladoras de gráficos: una placa de control principal con un procesador de 86 bits, un "Frame Grabber" y un convertidor de A/D de tres canales. El sistema resultante es capaz de producir animaciones, composición de imágenes y diseño gráfico hasta una capacidad total de composición tipográfica.

Frutos del éxito
Después del éxito en productos de ampliación para el ZX81, Memotech avanzó hacia nuevos productos. Uno es una línea de periféricos para ordenadores, como la impresora matricial de 80 columnas, la DMX 80. La última realización de Memotech es el MTX512, un pequeño microordenador personal y de oficina de 64 K. El MTX512 puede almacenar datos en cassette o en la unidad de disco flexible opcional que vemos en la fotografía

Cuando en 1982 Sinclair Research lanzó el Spectrum, Memotech decidió no producir ninguna gama de accesorios para la nueva máquina. En lugar de ello, utilizando la experiencia y la pericia obtenidas con la producción del hardware para el ZX81, Memotech optó por concentrar todos sus recursos en diseñar y construir sus propias máquinas. Tim Spencer, director de ventas y marketing de Memotech, explica: "Pensamos que el ZX81 no iba a durar mucho tiempo más, de modo que decidimos construir nuestro propio ordenador. Al fin y al cabo, teníamos la tecnología. Pero el ZX81 ha durado mucho tiempo más del que esperábamos, y nuestros paquetes se siguen vendiendo bien".

Memotech calcula que las ventas internacionales de sus Memopacks han sobrepasado las 250 000 unidades. Los paquetes, así como la gama de máquinas MTX, se fabrican en las oficinas centrales de la empresa, en Witney (Oxfordshire). En la actualidad el personal de la empresa sobrepasa los 100 empleados.

Preguntado sobre la filosofía de diseño de la gama MTX, Tim Spencer respondió: "Estamos dirigiéndonos al usuario personal más serio y al mercado de gestión. Las máquinas no están dirigidas al mercado de juegos aunque, por supuesto, con ellas uno puede practicar todos los juegos más conocidos".

Debido a que el MTX es capaz de ejecutar CP/M, puede aprovechar la gama de software disponible. Sin embargo, Memotech es consciente de una falta de software basado en cassette, que haría que la máquina le resultara más atractiva al usuario personal menos serio. En la actualidad sólo hay alrededor de 40 cassettes diferentes a la venta para el MTX, y la empresa está estimulando activamente el desarrollo de más programas. "Hemos hecho muchísimo durante estos últimos meses", nos comentaba Tim Spencer. "Tenemos estrechas relaciones con Continental Software, y PSS está escribiendo para nosotros." En el futuro cercano también habrá diversos paquetes educativos.

Gran final

Por último analizaremos algunos de los sistemas más avanzados con tecnología digital incorporada

De los recientes desarrollos en la música electrónica, el más significativo se ha producido en el área de la grabación digital. No sólo se ha mejorado enormemente la calidad de grabación del sonido, sino que el significado de la palabra "grabación" ha cambiado para abarcar diversidad de técnicas. Si entendemos que grabación significa "codificación digital del sonido y su organización en forma de música", podemos empezar a captar lo que está sucediendo con la música en la década de los ochenta.

Desde la segunda guerra mundial, la grabación de sonido en cinta magnética ha sido la norma, con formatos que van desde la más diminuta microcassette hasta los grandes carretes de cinta de 24 pistas utilizados en los estudios de grabación profesionales. Cuando se realiza una grabación en cinta magnética, las pequeñas partículas de óxido de metal de la superficie de la cinta se disponen en complejos patrones que son análogos a las formas de onda de sonido que representan. A medida que la cinta pasa por el cabezal de reproducción de un dispositivo de grabación, estos patrones se convierten en series de voltajes eléctricos. Estos voltajes entran posteriormente a los altavoces, los cuales reproducen el sonido grabado en la cinta.

Dado que la disposición de las partículas se puede conocer con bastante precisión en relación al cabezal de reproducción de cinta, es bastante sencillo descubrir dónde se encuentra un determinado sonido en un trozo de cinta magnética, de modo que el empalmado y la edición de cinta con hojas de afeitar desimantadas se ha convertido en una importante habilidad que los ingenieros de sonido deben adquirir.

En grabación digital, el sonido se codifica numéricamente a lo largo de la cinta, y el cabezal de reproducción se convierte en un convertidor de digital a analógico. La entrada a los altavoces es la misma que antes, pero éstos utilizan voltajes generados por el convertidor D/A. Siempre y cuando haya datos suficientes para convertir, la cinta digital logra una reproducción enormemente superior a la que se consigue con cinta magnética. Y, puesto que los datos permanecen intactos, la cinta se puede copiar digitalmente cientos de veces sin pérdida de calidad. Con la cinta magnética, sin embargo, cada "generación" de copias agrega "ruidos" y distorsión a la grabación.

Este tipo de degradación por ruido ha constituido un problema acuciante para los ingenieros de sonido durante muchos años. Ahora que se ha resuelto, muchos de los mejores estudios de graba-

Synclavier
El Synclavier, de New England Digital, se considera como uno de los más avanzados del mundo. Aparte de las funciones usuales de sintetizador, la máquina posee la capacidad de almacenar hasta 10 Megabytes de sonido en disco

Fairlight CMI
El Fairlight CMI fue uno de los primeros sistemas de música por ordenador. Su sistema operativo es conducido por menú, permitiendo diversas opciones, desde control por teclado a dibujo de formas de onda. Asimismo, la máquina tiene la facilidad de producir salidas impresas

Yamaha KX5
La interface MIDI del Yamaha KX5 proporciona un vínculo entre sintetizadores y el Yamaha CX5. Este dispositivo también se utilizará para conectar en interface el ordenador personal Yamaha MSX cuando éste se comercialice en Europa

Roland MSQ-700
El Roland MSQ-700 se presenta como el primer secuenciador del mundo compatible con la MIDI. El MSQ-700 posee una gama completa de facilidades MIDI y puede almacenar hasta 6 500 notas

Drumulator
La máquina de ritmos Drumulator, de Emu Systems, tiene una capacidad de memoria de 10 000 notas en 64 canciones. También hay facilidades para permitir la inserción de ROM adicionales que proporcionen efectos especiales, como percusión latina o africana

Ian McKinnell

ción han instalado grabadoras de cinta digitales de 24 pistas. Utilizándolas, la reproducción del sonido es tan exacta que a los ingenieros les resulta imposible distinguir si un sonido proveniente de los altavoces monitores del estudio lo está produciendo un músico en el área de grabación o si son fruto de la reproducción de una cinta digital. Pero han aparecido nuevos problemas: ya no se puede "ver" dónde estan situados los sonidos en una cinta digital, lo que hace que la edición resulte más difícil; por otra parte el empalmado se está convirtiendo en una habilidad obsoleta. Otra dificultad es el "ruido del estudio", una característica indeseable y a menudo también inaudible de algunos equipos de audio. La cinta magnética no era lo suficientemente sensible como para detectarlo, pero las grabaciones digitales sí tienden a captarlo.

Mientras que la grabación en 24 pistas sigue siendo todavía prerrogativa de los estudios caros, la grabación digital en una única pista, de la misma calidad, está a disposición de cualquier usuario de un video Betamax. La cinta de video es un soporte digital y, como tal, se puede utilizar para codificar cualquier tipo de datos. El Sony PCM (*Pulse Code Modulator*: modulador de códigos de impulsos) es una unidad que convierte a un video Betamax en una grabadora de cinta de audio. Esta unidad posee el potencial para hacer que las grabadoras analógicas de tamaño similar queden obsoletas.

La codificación digital del sonido, o muestreo (*sampling*), es el corazón del Fairlight CMI (Computer Musical Instrument), uno de los sistemas avanzados más conocidos. El Fairlight puede muestrear cualquier sonido de una duración de hasta dos segundos y reproducirlo luego a través de una escala de altura de seis octavas. El muestreo representa un avance decisivo en el campo de la música electrónica. Durante años, los ingenieros y los músicos han estado tratando de simular el sonido de instrumentos de cuerda o de viento y de madera utilizando sintetizadores, y en algunos casos han estado muy cerca de lograr su objetivo. Pero el muestreo no sólo proporciona una notable reproducción del sonido de cuerda, sino que puede producir el sonido de un violín en especial. Además, en algunos casos, puede reproducir el sonido de un determinado músico en una determinada habitación. En el primer capítulo de esta serie vimos cómo los compositores de *musique concrète* de los años cincuenta se pasaban semanas empalmando diminutos retazos de cinta grabada, produciendo finalmente obras a gran escala. Ahora la manipulación de muestras por ordenador permitirá que un compositor produzca resultados similares en cuestión de minutos.

Ecos del pasado
Este fragmento está extraído de *The new Atlantis* (La nueva Atlántida), una visión utópica escrita por el filósofo inglés Francis Bacon (1561-1626). Sus descripciones de los sonidos parecen anticipar el extraordinario poder y versatilidad de la música electrónica de hoy

Biblioteca Mary Evans

De LA NUEVA ATLÁNTIDA, de Francis Bacon, publicado en 1624:

También tenemos Casas de Sonido, donde practicamos y demostramos todos los sonidos, y su generación. Tenemos armonías, que ustedes no poseen, de cuartos de sonido, y de porciones de sonidos aún inferiores. Diversos instrumentos de música que ustedes desconocen igualmente, algunos de ellos más dulces que cualquiera que ustedes posean; junto con campanas y carillones que son dulces y exquisitos. Representamos pequeños sonidos como grandes y profundos, y los sonidos grandes debilitados y finos. Hacemos diversos temblores y sonidos de gorjeos, que en su original son enteros. Representamos e imitamos todos los sonidos articulados y las letras y las voces y las notas de bestias y pájaros. Contamos con ciertas ayudas, que colocadas junto a la oreja engrandecen cuanto se escucha. Asimismo tenemos diversos ecos extraños y artificiales, que reflejan la voz muchas veces y como si la estuvieran lanzando al aire. Y algunos que devuelven la voz más fuerte cuando llega, algunos más aguda y otros más profunda. Y algunos que dan la voz con letras o sonido articulado que difieren de la que reciben; también tenemos medios para transmitir sonidos por conductos y tubos, en extrañas líneas y distancias.

Instrumentos de muestreo

Un instrumento de muestreo como el Fairlight puede superar las limitaciones naturales de los instrumentos musicales. Por ejemplo, para un flautista es bastante fácil producir una calidad de tono cálida en el extremo inferior de la escala de la flauta. Sin embargo, para un músico, independientemente de lo hábil que sea, es imposible obtener este tipo de sonido dos octavas más arriba en el extremo superior de la escala del instrumento: el diseño físico de la flauta lo impide. Un usuario del Fairlight, por otra parte, puede muestrear el tono bajo y después transportarlo hacia arriba dos octavas en el teclado. El resultado seguirá sonando como una flauta, pero una clase de flauta que no puede existir en el mundo analógico "real".

El Fairlight puede suministrar una visualización en pantalla de cualquiera de los sonidos muestreados, que se almacenan en discos de 8″. Las diferentes características de un sonido individual pueden ser examinadas en sucesión; con frecuencia resulta más fácil decir qué es lo que está "mal" en un sonido determinado mirándolo en vez de escuchándolo. Muchos sonidos es necesario que sean más largos que su duración de muestra original de dos segundos. Viendo cómo están relacionadas las diferentes formas de onda dentro del sonido, se puede seleccionar un punto en el cual hacer que el sonido empiece a "buclar" (*looping*) o a repetirse. De seleccionarse el punto correcto, éste dará la ilusión de una auténtica continuidad. El sonido analógico no se puede comportar así, por supuesto, de modo que el "buclado" puede darle a la música que esté produciendo una dimensión insólita.

El usuario del Fairlight tiene dos formas de entrar la música, además de tocarla en "tiempo real" en el teclado. El primer método, conocido como *Page R* (página R), ofrece la visualización de un pentagrama, y el usuario entra las notas en éste desde el teclado de música. Cualquier error de tiempos lo "ordena" automáticamente el ordenador de acuerdo a la métrica o compás que el usuario haya especificado.

El segundo método consiste en utilizar un MCL (*Music Composition Language*: lenguaje para composición musical). El Fairlight MCL exige que cada evento de nota se entre mediante el teclado alfanumérico, pero permite modificar el tiempo y la acentuación de nota a nota. El Fairlight tiene una capacidad de ocho voces, de modo que un usuario podría entrar ocho frases distintas, tocadas por ocho sonidos o "instrumentos" distintos. Cada uno de éstos puede estar ligeramente fuera de tiempo (por apenas unos milisegundos) respecto a los otros, y toda la ejecución es coordinada por el reloj interno del Fairlight.

Quizá usted pregunte qué sentido tiene tocar música de esta forma "incorrecta", sobre todo cuando el ejecutante es un ordenador. La respuesta a esta cuestión es que las personas jamás tocaron exactamente al compás, y una de las características que definen la ejecución (especialmente entre los músicos de jazz y algunos clásicos) es la manera en la cual un músico puede modificar el tiempo musical cuando ejecuta un pasaje. Un sistema operativo como el Fairlight proporciona una forma en la cual se pueden simular ciertos estilos de ejecución. Estas simulaciones se pueden emplear en trabajos experimentales y de investigación, así como se utilizan las simulaciones por ordenador en el diseño de carrocerías de automóviles, de alas de aviones y de los escudos contra las altas temperaturas de las "lanzaderas espaciales" *(space shuttles)*.

Muchos músicos están justamente preocupados por la eventualidad de que instrumentos como el Fairlight lleguen a reemplazar a las personas, especialmente a medida que se vaya desarrollando la capacidad de simulación de tales equipos. Grupos como Wang Chung, Duran Duran y Culture Club utilizan Fairlight como parte de su proceso de producción, y con frecuencia resulta imposible decir qué es lo que realmente se está tocando y qué es lo que está ejecutando el Fairlight. No obstante, una vez que un usuario conoce el sistema operativo, se hace evidente que el Fairlight es mucho más que un mero instrumento musical nuevo y que tiene un potencial en gran medida todavía inexplorado.

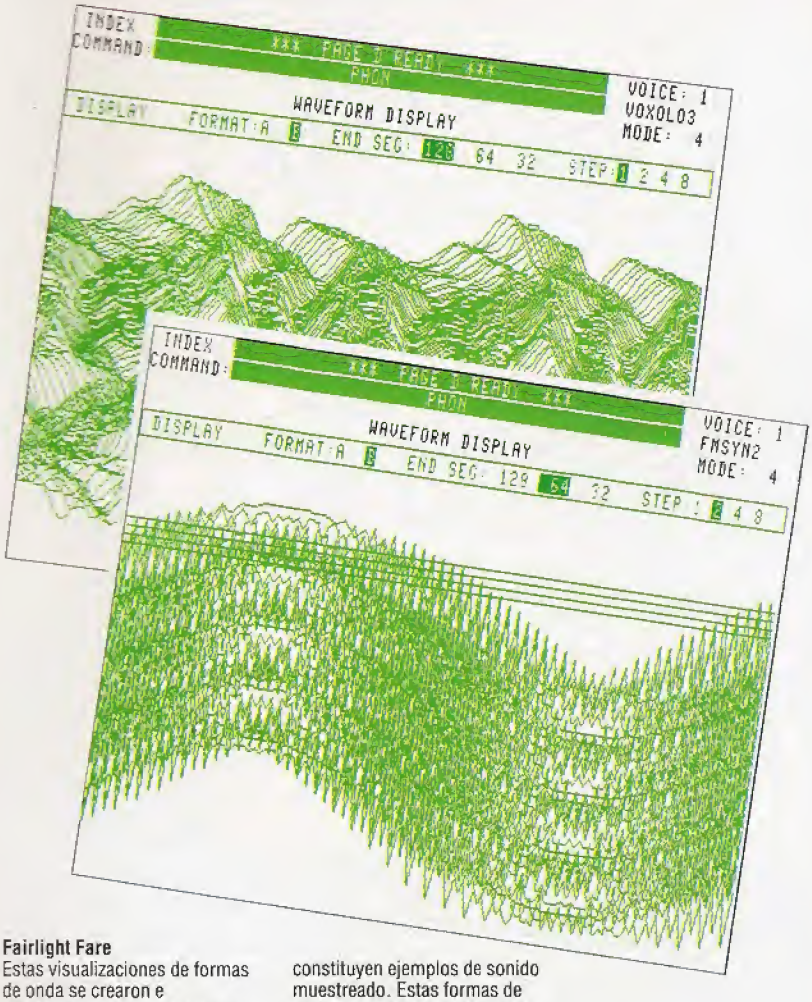
Si bien es el más conocido, el Fairlight no es el único instrumento de estas características que existe. El sistema Synclavier —que cuesta casi el doble que el Fairlight— posee facilidades similares, pero a mayor escala. Los datos se almacenan utilizando discos rígidos Winchester con una capacidad de 40 megabytes. Con el propio sistema Synclavier se podría producir y grabar un álbum completo, haciendo totalmente innecesaria una avanzada máquina digital de 24 pistas.

Pero hasta el Synclavier tiene sus limitaciones. En la actualidad todos los instrumentos de muestreo existentes poseen un rasgo común: reproducen el sonido muestreado de la forma más cercana posible al original, a menos que el usuario haya intervenido para realizar una modificación específica. Pero un trompetista, en medio de una actuación en vivo, puede introducir una considerable diferencia en el siguiente sonido a tocar simplemente cambiando el control de la respiración o la posición de los labios. Un trompetista competente hace esto prácticamente sin pensarlo. El siguiente paso para los instrumentos de muestreo sería, por consiguiente, producir un sistema "que responda al músico".

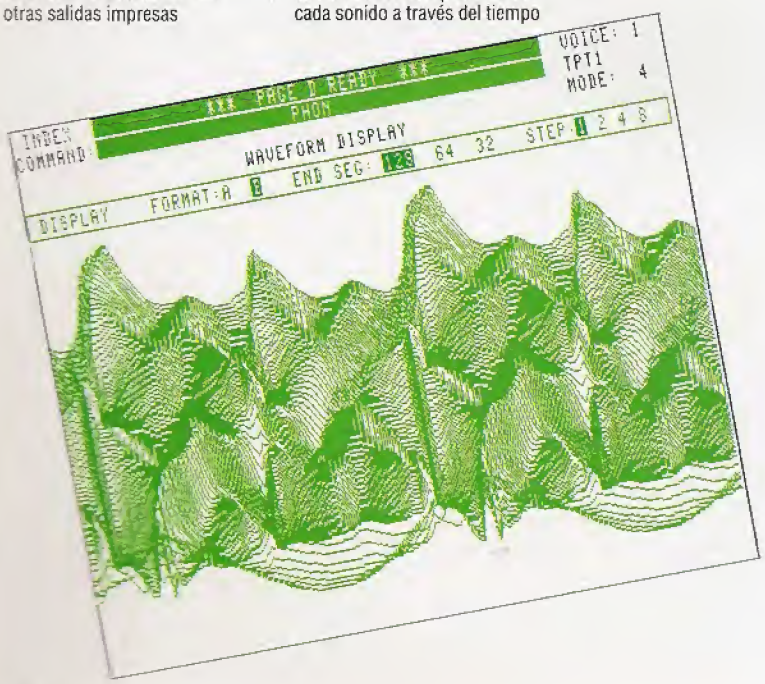
El Kurzweil, que aún es un sistema modélico, incorpora un programa de reconocimiento de patrones. Ello significa que cuando se toca una nota en el teclado de música, se exploran una cantidad de muestras diferentes y se combinan las características de cada muestra para producir el sonido individual. El tipo de características seleccionadas reflejará la forma en que se toque la música. Tal sistema se asemejaría cada vez más al carácter y la sensación de un auténtico instrumento acústico. La única diferencia es que ningún piano vertical ni piano de cola es exactamente igual a otro. Todos los "pianos" Kurzweil CMI serían idénticos en carácter y sensación, a menos que los usuarios desarrollen un método para escribir un software de su "propio carácter y sensación".

Se rumorea que Lucasfilm, productora de las películas *La guerra de las galaxias* e *Indiana Jones y el templo maldito*, está desarrollando un sistema aún más avanzado que el Fairlight, Synclavier y Kurzweil reunidos. Se espera que este sistema, denominado ASP (*Audio Signal Processor*: procesador de señales de audio), incorpore, en un solo sistema operativo complejo, todos los tipos de facilidades para sonido digital musical de que disponen actualmente sólo los grandes estudios musicales. De modo que, así como los ordenadores y los sintetizadores de los años cincuenta ocupaban el espacio equivalente a la superficie de varias habitaciones, y ahora ocupan sólo la superficie de un escritorio, podemos esperar que el estudio de grabación del futuro sea un paquete portátil.

Fairlight MCL
El lenguaje para composición musical utilizado por el Fairlight CMI es conducido por menú, de modo que las opciones se eligen a partir de menús claros y descriptivos que aparecen en la pantalla. Las formas de onda visualizadas en la página 1064 se generaron con una lista de parámetros de sonido en sentencias DATA, similar a un listado en BASIC, visualizándose e imprimiéndose luego utilizando instrucciones tomadas del menú



Fairlight Fare
Estas visualizaciones de formas de onda se crearon e imprimieron en un sistema de música Fairlight CMI. La primera es un ejemplo de patrón de onda sinusoide generado por el Fairlight utilizando síntesis FM (p. 1041). Este sonido se fabricó mezclando formas de onda electrónicamente. Las otras salidas impresas constituyen ejemplos de sonido muestreado. Estas formas de onda se producen digitalizando los sonidos verdaderos de una voz humana en el segundo caso, y de una trompeta en el tercero. Las visualizaciones son "tridimensionales" o topográficas, representando los cambios en la composición de cada sonido a través del tiempo



La incorporación de la tecnología digital no se ha limitado exclusivamente a los dispositivos y sistemas para generación de sonido. Los modernos estudios de grabación generalmente incluyen un cierto número de unidades para tratamiento de sonido como parte de su equipo básico. Un ejemplo es la *unidad de reverberación*. La música se encamina a través de esta unidad para agregar eco o "reverberación", y los guitarristas *rockabilly* y los productores de *dub reggae* han dependido de este tratamiento para darle a su música ese particular sonido.

El Quantec es una unidad digital que, en vez de meramente proporcionar reverberación, simula los tipos de reverberación que ocurren en habitaciones y espacios de diferentes dimensiones. Su "espacio acústico" más pequeño es una caja con un volumen de un metro cúbico, y las simulaciones preestablecidas incluyen dimensiones típicas de salas de estar, auditorios, hangares de aviones y catedrales. La característica más interesante del Quantec es la facilidad que ofrece para prolongar la reverberación bastante más allá de los límites acústicos o físicos naturales. De este modo, si se produce un sonido en la simulación de las dimensiones de catedral, y si se maximiza el tiempo de reverberación, todo el evento durará varios minutos: el efecto es algo así como escuchar un eco creado ¡por diez Grandes Cañones del Colorado unidos!

Se dice que en el Medio Oeste norteamericano de los años cincuenta había un pequeño estudio de *rockabilly* que se había construido cerca de un silo de cereales. Era este inmenso espacio el que le proporcionaba al estudio su clásico sonido *rockabilly* lleno de ecos. El empleo moderno de unidades digitales como el Quantec es más trivial, ciñéndose principalmente a las etapas de post-producción de los trabajos de cine y televisión. Se puede filmar a los actores hablando en el entorno acústicamente "muerto" de un estudio, y se puede tratar la banda de sonido después para proporcionar una resonancia acústica que se adapte al espacio en el cual se supone que transcurre la acción.

Valores "humanos"

Muchas personas, músicos y no músicos indistintamente, piensan que la tecnología digital aplicada a la música tiene un efecto pernicioso. Aducen que esta música está creada y tocada de una manera tan artificial que los auténticos valores "humanos" de la espontaneidad y la expresividad quedan sepultados en la carrera por adquirir niveles de control técnico cada vez mayores. Éste es un argumento convincente, pero vale la pena considerar un medio visual como el cine para juzgar si es éste realmente el caso.

Durante varias décadas la tecnología cinematográfica les ha permitido a los directores rodar escenas desde cualquier ángulo, acercar la cámara para filmar con detalle y girarla para captar grandes panorámicas. Ha sido posible repetir secuencias, ralentizarlas, acelerarlas, hacerlas retroceder y montar la película de modo que en fracciones de segundo puedan ocurrir las yuxtaposiciones visuales más insólitas. Sólo recientemente, con la tecnología digital, los compositores de música han comenzado a ejercer sobre el sonido un control comparable, de modo que es posible que la música digital finalmente se considere bajo el mismo prisma.

Control de motores

Ahora veremos qué software se requiere para conmutar motores y desarrollaremos un sistema sencillo de control de realimentación

El control por microprocesador de dispositivos externos actualmente es común en la industria, con aplicaciones que van desde contar las botellas que pasan por una cinta transportadora hasta soldar carrocerías de automóviles. Los principios de todos los sistemas de control son, en primer lugar, entrar datos desde el mundo exterior en un formato comprensible para un sistema de microprocesador; en segundo término analizar estos datos, y, finalmente, instigar acciones externas basadas en ese análisis. Si estas tres actividades se repiten en un ciclo continuo, entonces tenemos un sistema que se conoce como *control de realimentación* o de *feedback*.

Para ilustrar el principio del control de realimentación tomemos el ejemplo de calentar una cacerola con sopa en una cocina. Con el fin de cocer la sopa, el calor suministrado debe ser suficiente para hacerla hervir, pero no tanto como para que se derrame por los bordes de la cacerola. Si fuéramos a llevar a cabo esta tarea nosotros mismos, inicialmente aplicaríamos calor máximo a la cacerola hasta que la sopa comenzara a hervir y entonces lo reduciríamos hasta dejar el líquido hirviendo suavemente. Si en cualquier momento la sopa comenzara a hervir demasiado, reduciríamos aún más el calor. Al hacer esta clase de operación nosotros controlamos visualmente el estado de la sopa, analizamos lo que vemos y emprendemos la acción apropiada. Repetiríamos estas acciones hasta que el alimento estuviera listo para ser ingerido. Un microordenador controlaría la cocción de la sopa de una forma similar, aunque no idéntica. La principal diferencia radicaría en la forma de controlar el estado de la sopa. Mientras que nosotros podemos mirarla y formular un juicio valiéndonos de nuestra experiencia, un sistema informático tendría que utilizar otro método basado en propiedades físicas fácilmente determinables, tales como la temperatura. Antes de que se informatizara el control de la sopa, alguien debería llevar a cabo experimentos iniciales para determinar la temperatura que le corresponde a una suave ebullición constante y a partir de qué temperatura la sopa comenzará a hervir peligrosamente. A partir de entonces, sin embargo, el ordenador asumirá la tarea, siempre y cuando estén disponibles los dispositivos apropiados que permitan medir la temperatura y regular el calor.

Se pueden controlar motores de varias formas utilizando la caja buffer (véase p. 1026) y la caja de salida de bajo voltaje (véase p. 1054) que hemos construido. Un motor de juego de tren eléctrico o Lego es ideal para comprobar el software que diseñaremos. Sólo necesitamos asegurar que el motor que conectemos a la caja de salida tenga un voltaje nominal igual o mayor que el voltaje de entrada del transformador. Conectando los dos terminales del motor a la línea 0 de la caja de salida, podemos encender y apagar el motor utilizando las teclas "Z" y "X" del teclado.

BBC MICRO

```
10 REM MOTOR SENCILLO BBC
20 RDD=&FE62:REGDAT=&FE60
30 ?RDD=255:REM TODAS LAS LINEAS SALIDA
40 ?REGDAT=0:REM MOTOR APAGADO
50 REPEAT
60 A$=INKEY$(1):REM PULSACION DE TECLA?
70 IF A$="Z"THEN ?REGDAT=1: REM ENCENDER
80 UNTIL A$="X"
90 ?REGDAT=0:REM APAGAR
```

COMMODORE 64

```
10 REM MOTOR SENCILLO CBM64
20 RDD=56579:REGDAT=56577
30 POKERDD,255:REM TODAS LAS LINEAS SALIDA
40 POKEREGDAT,0:REM MOTOR APAGADO
50 GETA$:IFA$ <> "Z" ANDA$ <> "X" THEN
   50:REM ESPERAR PULSACION TECLA
60 IF A$="Z" THEN POKEREGDAT,1:GOTO50
70 IF A$="X" THEN POKEREGDAT,0:END
```

Ejercicios

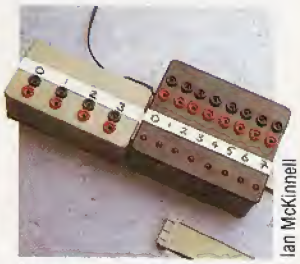
Ahora estamos en condiciones de diseñar algunos elegantes experimentos de control que se basan en entrada y salida utilizando la caja buffer y la caja de salida de bajo voltaje. Los sensores a los que hacen referencia pueden ser sensores de presión o interruptores "de lengüeta" operados por un pequeño imán. Tales dispositivos se pueden conseguir en tiendas de electricidad y electrónica a un precio muy reducido. Los interruptores sensibles al calor operan haciendo un contacto interno entre los terminales cuando se alcanza una cierta temperatura.

1) Escriba un programa para hacer que un vehículo corra hacia atrás y hacia adelante entre dos sensores colocados en su camino.

2) Escriba un programa para hacer que un vehículo se detenga justo encima de un sensor, haciendo marcha atrás de ser necesario.

3) Escriba un programa para mantener una cubeta de agua entre 70 y 100 °C usando un calentador de bajo voltaje y dos interruptores sensibles al calor.

4) Escriba un programa para calcular la velocidad en metros por segundo de un vehículo que viaje entre dos puntos. (Necesitará conocer la distancia y registrar el tiempo invertido en recorrerla.)



Ian McKinnell

Sacando más buffers
La caja buffer, nuestra primera construcción, está destinada a proteger la circuitería del ordenador contra corrientes de entrada o salida excesivas. Su fuente de alimentación eléctrica independiente activa la caja de salida (el proyecto más reciente) y permite la conmutación controlada por software de la salida de 12 voltios

Podemos controlar la dirección del motor conectando sus terminales a líneas adyacentes de la caja de salida. El diagrama ilustra estas conexiones. Y utilizaremos un interruptor simple conectado a la línea 7 de la caja buffer para controlar la dirección.

En el siguiente programa, un valor de 1 en el registro de datos hace que fluya corriente en un sentido a través del motor. Colocando un valor de 2 en el registro de datos, la corriente fluirá en el sentido contrario. El programa investiga repetidamente la línea 7 y sólo coloca un 2 en el registro de datos cuando la línea está baja (es decir, el interruptor está cerrado). De esta forma, el cierre y la apertura del interruptor controlan la dirección del motor. Éste es un ejemplo muy sencillo de un sistema de control de realimentación.

BBC MICRO

```
10 REM MOTORES DIRIGIDOS BBC
20 RDD=&FE62:REGDAT=&FE60
30 ?RDD=127:REM LINEA 7 ENTRADA
40 ?REGDAT=0:REM APAGAR
50 A$=GET$:REM ESPERAR PULSACION TECLA
60 REPEAT
70 A$=INKEY$(1)
80 IF(?REGDAT AND 128)=0 THEN DIR=
   2 ELSE DIR=1
90 ?REGDAT=DIR
100 UNTIL A$="X":REM PULSAR X PARA ACABAR
110 ?REGDAT=0:REM APAGAR
```

COMMODORE 64

```
10 REM MOTORES DIRIGIDOS CBM64
20 RDD=56579:REGDAT=56577
30 POKERDD,127:REM LINEA 7 ENTRADA
40 POKEREGDAT,0:REM TODO APAGADO
50 GETA$:IFA$=""" THEN50:REM ESPERAR
   PULSACION TECLA
60 GETA$
70 IF(PEEK(REGDAT)AND128)=0THENPOKEREGDAT,
   2:GOTO90
80 POKEREGDAT,1
90 IFA$ <> "X"THEN60
100 POKEREGDAT,0:REM APAGAR
```

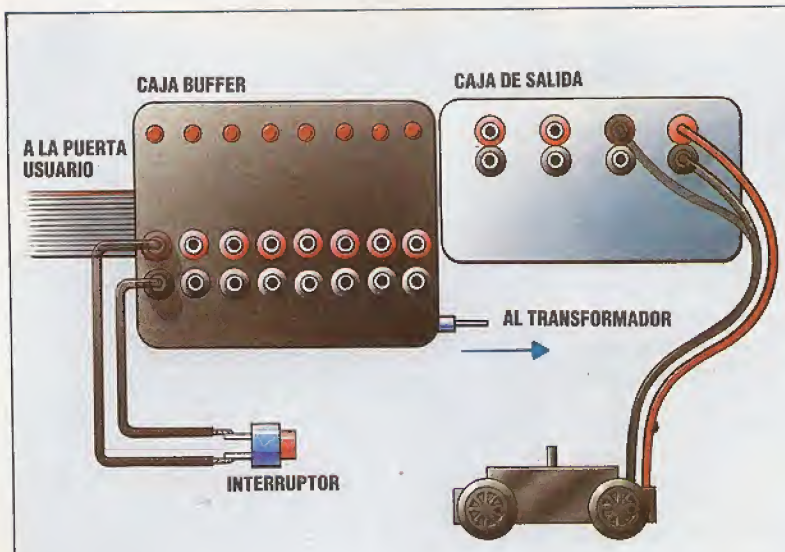
Además de controlar la dirección de un motor, también podemos controlar su velocidad directamente desde la caja de salida. Ello no exige dispositivos complicados, como convertidores de digital a analógico para controlar el suministro de corriente a los motores. En cambio, podemos enviarle impulsos al motor, encendiéndolo y apagándolo en rápida sucesión. Si hacemos esto con la suficiente rapidez, parecerá que éste gira continuamente: el intervalo entre cada impulso determinará la velocidad a la cual gira. Para programar esto sólo es necesario un par de bucles de demora de longitud regulable, dentro de una estructura repetitiva más grande, para determinar el período de tiempo en el que el motor está encendido o apagado durante cada ciclo.

BBC MICRO

```
10 REM CONTROL DE MOTOR VARIABLE BBC
20 RDD=&FE62:REGDAT=&FE60: VELOCIDAD=30
30 ?RDD=255:REM TODAS SALIDA
40 ?REGDAT=0:REM TODO APAGADO
50 A$=GET$:REM ESPERAR PULSACION DE TECLA
60 REPEAT
70 A$=INKEY$(1)
80 ?REGDAT=0:REM APAGAR
90 FORI=1TO(100-VELOCIDAD):NEXT:REM
   DEMORA1
100 ?REGDAT=1:REM ENCENDER
110 FORI=1TOVELOCIDAD:NEXT:REM DEMORA2
120 IF A$="D" THEN VELOCIDAD=VELOCIDAD-5
130 IF A$="Z" THEN VELOCIDAD=VELOCIDAD+5
140 UNTIL A$="X"
150 ?REGDAT=0:REM APAGAR
```

COMMODORE 64

```
10 REM CONTROL DE MOTOR VARIABLE CBM64
20 RDD=56579:REGDAT=56577:VELOCIDAD=30
30 POKERDD,255:REM TODAS LAS LINEAS SALIDA
40 POKEREGDAT,0:REM APAGAR
50 GETA$:IFA$=""THEN50:REM ESPERAR
   PULSACION TECLA
60 GETA$
70 POKEREGDAT=0:REM APAGAR
80 FORI=1TO(100-VELOCIDAD):NEXT:REM
   DEMORA1
```



Paseo sobre cuatro ruedas
Las dos cajas comparten el bus de datos y la fuente de alimentación eléctrica a través de sus puertas minicon; si cada caja se construye con un conector y un enchufe, entonces se pueden conectar cajas en cadena.
El coche tiene un motor de 12 v de continua, suministrados desde la caja de salida y conmutado mediante el bit 7 de la caja buffer. La polaridad de las salidas se pueden invertir por software, activando el coche hacia atrás y hacia adelante

Kevin Jones

```
90 POKEREGDAT=1:REM ENCENDER
100 FORI=1TOVELOCIDAD:NEXT:REM DEMORA2
110 IFA$="D"THENVELOCIDAD=VELOCIDAD-5
120 IFA$="Z"THENVELOCIDAD=VELOCIDAD+5
130 IFA$ <> "X"THEN60
140 POKEREGDAT,0:REM APAGAR
```

En este programa se utiliza la variable VELOCIDAD para determinar la longitud de cada bucle de demora. La lógica del bucle es tal que a medida que una demora aumenta, la otra disminuye, y viceversa. DEMORA1 determina el período en el cual el motor está apagado, y DEMORA2 el período en el cual está encendido. Para grandes valores de VELOCIDAD, la primera demora es breve y la segunda es larga, haciendo que el motor gire más rápidamente. Valores más pequeños de VELOCIDAD producirán períodos más largos con el motor apagado durante cada ciclo, haciendo que parezca que gire más lentamente. El efecto de impulso del programa se puede observar en el parpadeo del LED de la línea 1.

Diversión y aprendizaje
Conducir coches de juguete guiados por cable no parece ser una gran compensación para el esfuerzo y el capital invertidos, pero la construcción y programación de incluso estos artefactos tan simples nos ha introducido en la realidad de la construcción electrónica y electromecánica y nos ha mostrado algunos de los problemas a resolver para lograr la interacción entre el software y el mundo real

Ian McKinnell

Respuestas a los ejercicios

1) Suponiendo que los sensores están conectados a las líneas 6 y 7, y que el motor está conectado entre los terminales positivos de las líneas 0 y 1:

```
10 REM BBC VERSION 3.1
20 RDD=&FE62:REGDAT=&FE60
30 ?RDD=63:REM LINEAS 6 & 7 ENTRADA
40 adelante=1:atras=2
50 ?REGDAT=adelante
60 FORI=1TO2000:NEXT:REM DEMORA
70 REPEAT UNTIL(?REGDAT AND192) <> 192
80 ?REGDAT=atras
90 FORI=1TO2000:NEXT:REM DEMORA
100 IF(?REGDAT AND192) <> 192 THEN50 ELSE
    GOTO100

10 REM CBM 64 VERSION 3.1
20 RDD=56579:REGDAT=56577
30 POKERDD,63:REM LINEAS 6&7 ENTRADA
40 AD=1:AT=2
50 POKE REGDAT,AD
60 FORI=1TO1000:NEXT:REM DEMORA
70 IF(PEEK(REGDAT)AND192)=192THEN70
80 POKE REGDAT,AT
90 FORI=1TO1000:NEXT:REM DEMORA
100 IF(PEEK(REGDAT)AND192) <> 192THEN50
110 GOTO100
```

2) Suponiendo que el sensor se conecta a la línea 7 y el motor se conecta entre las líneas 0 y 1:

```
10 REM BBC VERSION 3.2
20 RDD=&FE62:REGDAT=&FE60
30 ?RDD=127:REM LINEA 7 ENTRADA
40 velocidad=30:adelante=1:atras=2
50 direccion=adelante
60 REPEAT
70 ?REGDAT=0:REM APAGADO
80 FOR I=1TO(100-velocidad):NEXT
90 ?REGDAT=direccion
100 FOR I=1TOvelocidad:NEXT
110 UNTIL(?REGDAT AND128)=0:REM INTERRUPTOR
120 FOR I=1TO1000:NEXT:REM DEMORA
130 REM COMPROBAR SI EXCEDIDO
140 IF(?REGDAT AND128)=0THEN?REGDAT=0:END
150 REM ATRAS LENTAMENTE
160 velocidad=2:direccion=atras:GOTO60

10 REM CBM64 VERSION 3.2
20 RDD=56579:REGDAT=56577
30 POKE RDD,127:REM LINEA 7 ENTRADA
40 VEL=30:AD=1: AT=2
50 DR=AD
60 POKE REGDAT,0:REM APAGADO
70 FOR I=1TO(100-VEL):NEXT
80 POKE REGDAT,DR
90 FORI=1TOVEL:NEXT
100 IF(PEEK(REGDAT)AND128) <> 0THEN60
110 FORI=1TO1000:NEXT:REM DEMORA
120 IF(PEEK(REGDAT)AND128)=0THENPOKE
    REGDAT,0:END
130 REM HACIA ATRAS LENTAMENTE
140 VEL=2:DR=AT:GOTO60
```

3) Suponiendo que los sensores de 40 y 70° están conectados a las líneas 6 y 7 respectivamente, y que el calentador está fijado a la línea 0:

```
10 REM BBC VERSION 3.3
20 RDD=&FE62:REGDAT=&FE60
30 ?RDD=63:REM LINEAS 6&7 ENTRADA
40 REPEAT
50 A$=INKEY$(1)
60 ?REGDAT=1:REM ENCENDER CALENTADOR
70 REPEAT
80 UNTIL(?REGDAT AND192)=0:REM 70 GR
90 ?REGDAT=0:REM APAGAR CALENTADOR
100 REPEAT UNTIL(?REGDAT AND192)=192
110 UNTIL A$ <> "":REM PULSACION TECLA

10 REM CBM64 VERSION 3.3
20 RDD=56579:REGDAT=56577
30 POKE RDD,63:REM LINEAS 6&7 ENTRADA
40 GET A$
50 ?REGDAT=1:REM ENCENDER CALENTADOR
60 IF(PEEK(REGDAT)AND192) <> 0THEN60
70 POKE REGDAT,0:REM APAGAR CALENTADOR
80 IF(PEEK(REGDAT)AND192) <> 192THEN80
90 IF A$=""THEN40
```

4) Suponiendo que el primer interruptor está en la línea 6 y el segundo en la línea 7:

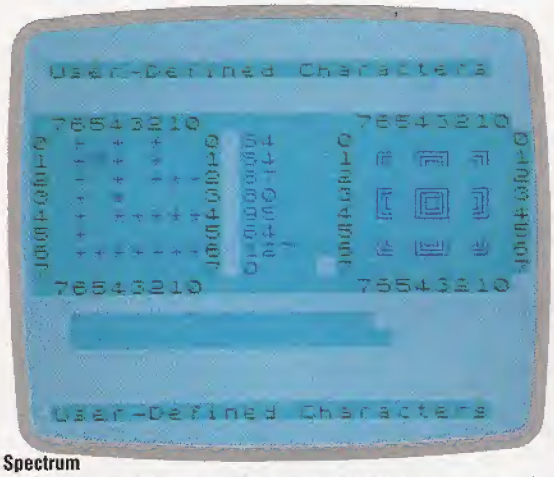
```
10 REM BBC VERSION 3.4
20 RDD=&FE62:REGDAT=&FE60:DISTANCIA=1
30 ?RDD=63
40 REPEAT UNTIL(?REGDAT AND64)=0
50 ?REGDAT=1
60 TIEMPO=0:REM INICIALIZAR TEMPORIZADOR
70 REPEAT UNTIL(?REGDAT AND128)=0
80 PRINT"VELOCIDAD="DISTANCIA/(TIEMPO/100)
90 ?REGDAT=0

10 REM CBM64 VERSION 3.3
20 RDD=56579:REGDAT=56577:DS=1
30 POKE RDD,63
40 IF(PEEK(REGDAT)AND64) <> 0THEN40
50 POKE REGDAT,1
60 T=TI:REM INICIALIZAR TEMPORIZADOR
70 IF(PEEK(REGDAT)AND128) <> 0THEN60
80 PRINT"VELOCIDAD="DS/((TI-T)/60)
90 POKE REGDAT,0
```

Formación de caracteres

A través de dos programas le mostramos con qué facilidad se pueden crear caracteres en el Spectrum y el BBC Micro

El enfoque del Spectrum a los caracteres definidos por el usuario es muy práctico: hay 168 bytes en el tope de la memoria dedicados por el sistema operativo a las definiciones de caracteres del usuario (si bien esta memoria se puede utilizar para otros fines). Este espacio permite almacenar 21 definiciones de caracteres, y el sistema operativo le adjunta a cada definición un valor CHR$ en la escala entre 144 y 164. Los caracteres así definidos son considerados por la máquina como los caracteres de la "a" a la "u" en modalidad de gráficos (G). La variable del sistema (dirección 32600) para gráficos definidos por el usuario (UDG) apunta al primer byte de la zona de memoria dedicada, pero no es necesario efectuar sumas complicadas con esta variable para hallar la dirección de comienzo de una definición de carácter. La instrucción LET DIRECCION=USR"A" devuelve la dirección del primer byte de la definición del carácter contenido entre las comillas.

Comparando la longitud y la estructura de los listados para el Commodore y el Spectrum se aprecia la sencillez de los métodos de este último. El programa es una traducción de la versión para el Commodore (p. 1052). Las teclas del cursor normales del Spectrum (SHIFT 5, SHIFT 6, etc.) controlan el cursor de edición, y las teclas 6, 7 y 8 sin SHIFT son de instrucciones (activar una celda, cambiar el carácter editado y colocar un carácter en la ventana de texto). El signo de exclamación es la instrucción de salida.



Spectrum

Una vez definidos los nuevos caracteres, puede guardarse (SAVE) el área UDG con la instrucción:

```
SAVE"archudg" CODE(USR"A"),168
```

y volver a cargarla con:

```
LOAD"archudg"CODE
```

El tratamiento del BBC de los gráficos definidos por el usuario es similar, a primera vista, al del Spectrum. Los 256 bytes entre &0C00 y &0CFF están reservados para las definiciones de los caracteres codificados en ASCII del 224 al 255. Si el juego de caracteres se implosiona (véase *Guía para el usuario avanzado del BBC*, p. 136), estas definiciones también se aplican a los códigos ASCII del 128 al 159, del 160 al 191 y del 192 al 223. En el estado explosionado, todo el juego de caracteres imprimibles (desde CHR$(32) hasta CHR$(255)) puede ser redefinido, pero al costo de RAM del usuario. Para la mayoría de los fines, debería ser suficiente un máximo de 32 caracteres especiales.

Las funciones del programa son las mismas que en las versiones para el Commodore y el Spectrum. Las teclas de flecha controlan el cursor, y las teclas de función, de f1 a f3, las instrucciones para activar, redefinir y colocar. Como antes, el signo de exclamación, "!", es el terminador del programa.

BBC Micro

Ian McKinnell

Observe las llamadas al sistema operativo de las líneas 70 a 180: activan y desactivan las funciones COPY de modo que se puedan utilizar en el programa las teclas de flecha; si se sale del programa utilizando un modo diferente a la instrucción "!", debe digitarse *FX4,0 al comienzo de una nueva línea para reactivar las teclas de edición.

El BBC Micro permite emplear VDU23 para definir caracteres, pero en este caso es más simple calcular las direcciones pertinentes y después cogerlas (PEEK) y colocarlas (POKE) individualmente.

Puede salvar (SAVE) sus caracteres especiales así:

```
*SAVE"nombrearchivo"0C00 0CFF
```

y volver a cargarlos con:

```
*LOAD""
```

Spectrum

```
 19 REM ********************************
 20 REM*                spectrum         *
 21 REM *gen.de car.def.usuario*
 22 REM ********************************
100 GO SUB 1000: REM inicializar
110 FOR j=0 TO 1 STEP 0
120 GO SUB 2500: REM entrada
130 GO SUB 3000: REM validar
140 GO SUB apuntador:REM obedecer
150 NEXT j
200 STOP
999 REM ********************************
1000 REM *              inicializar        *
1001 REM ********************************
1020 DIM b(8,8): DIM c$(2,2): DIM o(2,7): DIM
     r$(8): DIM d$(1): DIM t(8,8)
1100 LET teclanula=2000: LET movercrsr
     =3500: LET instruccion=3000: LET
     actualizar=6500
1200 REM ********inicial.pantalla ********
1220 LET r0=4: LET c0=3: LET r1=8: LET
     c1=8: LET of=16
1230 LET negro=0: LET azul=1: LET cyan=5:
     LET blanco=7
1240 PAPER cyan: INK negro
1250 LET z$="Caracteres definidos por el
     usuario":PRINT AT 1,4;z$; PRINT AT
     20,4;z$
1260 LET z$="  76543210": PRINT AT
     r0,c0;z$;: PRINT TAB c0+of;z$
1270 PRINT AT r0+c1+1,c0;z$;: PRINT TAB
     c0+of;z$
1280 FOR r=r0+1 TO r0+c1
1290 LET z$=STR$ (r-r0-1): LET
     z$=z$+"       "+z$
1300 PRINT AT r,c0;z$;AT r,c0+of;z$
1310 NEXT r
1420 REM ********* despl. cursor *********
1440 DATA -1,+1,0,0
1445 DATA 0,0,+1,-1
1450 DATA 0,0,+1,-1
1460 FOR k=1 TO 2:FOR i=1 TO 4
1470 READ o(k,1):NEXT I:NEXT k
1600 REM ******* inicial.ventana texto*******
1620 FOR r=1 TO r1
1640 FOR c=1 TO c1
1660 LET t(r,c)=32
1680 NEXT c:NEXT r
1800 LET rpos=1: LET cpos=1:LET cn=144:
     LET e$="A"
1820 GO SUB 6000
1990 RETURN
1999 REM ********************************
2000 REM *       pulsacion tecla invalida     *
2001 REM ********************************
2020 BEEP .2,-3: RETURN
2120 RETURN
2499 REM ********************************
2500 REM *            crsr @,rpos,cpos        *
2501 REM ********************************
2520 LET rp=r0+rpos: LET cp=c0+cpos: LET
     cf=1+b(rpos,cpos): LET d$=" "(cf)
2540 PRINT AT rp,cp; FLASH 1;d$
2600 REM LET t$=INKEY$:IF t$ <> "" THEN
     GO TO 2600
2620 LET t$=INKEY$: IF t$="" THEN GO TO
     2620
2640 PRINT AT rp,cp;FLASH 0;d$
2700 RETURN
2999 REM ********************************
3000 REM *             validar entrada          *
3001 REM ********************************
3020 IF t$="!" THEN LET apuntador=teclanula:
     LET j=2: RETURN
3040 LET tecla=CODE (t$)-7: LET k=tecla-45:
     LET apuntador=teclanula
3060 IF (tecla > =1 AND tecla < =4) THEN
     LET apuntador=movercrsr: RETURN
3080 IF(t$ > "6" AND t$ < ="8") THEN LET
     tecla=VAL (t$)-4: LET apuntador=
     instrucción+tecla*500: RETURN
3100 RETURN
3499 REM ********************************
3500 REM *             mover el cursor          *
3501 REM ********************************
3520 LET ny=rpos+o(2,tecla): LET
     nx=cpos+o(1,tecla)
3540 IF (ny < 1 OR ny > r1) THEN RETURN
3560 IF (nx < 1 OR nx >r1) THEN RETURN
3580 LET rpos=ny: LET cpos=nx
3600 RETURN
3999 REM ********************************
4000 REM *            activar una celda         *
4001 REM ********************************
4020 LET tg=1-b(rpos,cpos): LET
     d$=" "(1+tg)
4040 PRINT AT rpos+r0,cpos+c0;d$
4060 LET b(ros,cpos)=tg
4120 LET pe=PEEK (mpos+rpos)
4140 LET pe=pe+(tg*2-1)*(2^(8-cpos))
4160 POKE (mpos+rpos),pe
4200 PRINT AT r0+rpos,c0+c1+3;PEEK
     (mpos+rpos);" "
4220 NEXT r
4300 RETURN
4499 REM ********************************
4500 REM *          definir nuevo caracter      *
4501 REM ********************************
4520 PRINT AT r0+r1+r3,c0+2;FLASH
     1;"Cambiar caracter"
4530 GO SUB actualizar
4540 PRINT AT r0+r1+4,c0+2; "Pulse tecla
     (a-u)"
4550 FOR z=1 TO 1
4560 LET e$=INKEY$: IF e$ <>"" THEN GO TO
     4560
4570 LET e$=INKEY$: IF e$="" THEN GO TO
     4570
4580 IF (e$ < "a" OR e$ > "u") THEN GO SUB
     teclanucla: LET z=0
4590 NEXT z
4600 LET cn=CODE (e$)+79
4610 IF cn > 164 THEN LET cn=cn-32
4620 PRINT AT r0+r1+3,c0+2;FLASH
     0;"                "
4630 PRINT AT
     r0+r1+4,c0+2;"
4640 GO SUB 6000
4990 RETURN
4999 REM ********************************
5000 REM *            situar un caracter        *
5001 REM ********************************
5020 PRINT AT rpos+r0,cpos+c0+of;CHR$ (cn)
5040 LET t(rpos,cpos)=cn
5490 RETURN
5999 REM ********************************
6000 REM *           visualizar un car.         *
6001 REM ********************************
6020 LET mpos=USR (e$)-1: INK azul
6040 FOR r=1 TO c1: LET pe=PEEK(mpos+r):
     LET p=pe: LET z$=""
6060 FOR c=c1 TO 1 STEP -1: LET n=INT
     (pe/2): LET q=pe-2*n
6080 LET b(r,c)=q: LET pe=n
6100 LET r$(c)="*" (q+1): NEXT c
6120 PRINT AT r0+r,c0+1;r$;AT
     r0+r,c0+c1+3;p;" "
6130 NEXT r
6140 RETURN
6499 REM ********************************
6500 REM *          actualizar ventana texto    *
6501 REM ********************************
6520 FOR r=1 TO 8: LET y=r0+r
6540 FOR c=1 TO 8: LET x=c0+c+of
6560 PRINT AT y,x;CHR$ (t(r,c))
6580 NEXT c:NEXT r
6600 RETURN
```

BBC Micro

```
 19 REM ********************************
 20 REM *              BBC Micro           *
 21 REM ********************************
 22 REM *     GEN.CAR.DEF.POR USUARIO      *
 50 MODE 1
 60 @ %=3
 70 *FX4,1
100 PROCinicializar
110 REPEAT
120 PROCtomar—instruccion
130 PROCvalidar—instruccion
140 PROCobedecer(INSTRUCCION)
160 UNTIL INSTRUCCION=SALIR
180 *FX4,0
200 END
999 REM ********************************
1000 DEPROCinicializar
1001 REM ********************************
1040 DIM BD(8,8),C$(2,2)
1045 DIMOFST(2,7),D$(1),TX(8,8)
1060 D$(0)="  ":D$(1)="**"
1080 CGEN=&DC00-1
1100 SALIR=-1:TECLANULA=0:MVRCRSR=1
1105 TGGLE=2:DFINIR=3:COLOCAR=4
1200 REM ***** INICIALIZAR PANTALLA ******
1220 R0=4:C0=3:RL=8:CL=8
1225 C9=CL+3:OF=16
1230 NEGRO=0:ROJO=1:AMARILLO=2
1235 BLANCO=3:FONDO=128
1240 COLOUR FONDO+ROJO
1245 COLOUR AMARILLO:CLS
1250 Z$="CARACT. DEF. POR USUARIO"
1255 PRINTTAB(4,1)Z$;TAB(4,20)Z$
1260 Z$="  76543210"
1264 PRINTTAB(C0,R0)Z$
1268 PRINTTAB(C0+OF,R0)Z$
1270 Y=R0+RL+1:X=C0
1274 PRINTTAB(X,Y)Z$
1278 PRINTTAB(X+OF,Y)Z$
1280 FOR R=R0+1 TO R0+RL
1290 Z$=STR$(R-R0-1)
1295 Z$=Z$+"       "+Z$
1300 PRINTTAB(C0,R)Z$
1304 PRINTTAB(C0+OF,R)Z$
1310 NEXT R
1320 COLOUR BLANCO
1420 REM ******** DESPL. CURSOR *********
1440 DATA -1,+1,0,0
1450 DATA 0,0,+1,-1
1460 FOR K=1 TO 2:FOR L=1 TO 4
1470 READ OFST(K,L):NEXT L,K
1500 REM ** ESTABLECIMIENTO DE TECLAS ***
1520 *TECLA1"T"
1540 *TECLA2"D"
1560 *TECLA3"P"
1600 REM ***** INICIAL. VENTANA TEXTO *****
1620 FOR R=1 TO RL
1640 FOR C=1 TO RL
1660 TX(R,C)=32
1680 NEXT C,R
1800 RPOS=1:CPOS=1:CN=224
1805 PROCvisualizar—caracter
1990 ENDPROC
1999 REM ********************************
2000 DEFPROCteclanula
2001 REM ********************************
2020 SOUND 1,-15,48,10
2120 ENDPROC
2499 REM ********************************
2500 DEFPROC tomar—instruccion
2501 REM ********************************
2520 RP=R0+RPOS:CP=C0+CPOS
2540 PRINTTAB(CP,RP);
2620 T$=GET$
2700 ENDPROC
2999 REM ********************************
3000 DEFPROC validar—instruccion
3001 REM ********************************
3020 INSTRUCCION=MVRCRSR
3040 TECLA=ASC(T$)-135
3060 IF TECLA < 1 THEN
     INSTRUCCION=TECLANULA
3065 IF TECLA > 4 THEN
     INSTRUCCION=TECLANULA
3080 IF T$="T" THEN INSTRUCCION=TGGLE
3100 IF T$="D" THEN INSTRUCCION=DFINIR
3120 IF T$="P" THEN INSTRUCCION=COLOCAR
3150 IF T$="!" THEN INSTRUCCION=SALIR
3160 ENDPROC
3299 REM ********************************
3300 DEFPROCobedecer(instruccion)
3301 REM ********************************
3320 IF instruccion=SALIR THEN PROCteclanula
3340 IF instruccion=MVRCRSR THEN
     PROCmovercursor(TECLA)
3360 IF instruccion=TGGLE THEN
     PROCactivar—una—celda
3380 IF instruccion=DFINIR THEN
     PROCdefinir—caracter
3400 IF instruccion=COLOCAR THEN
     PROCcolocar—caracter
3420 IF instruccion=TECLANULA THEN
     PROCteclanula
3450 ENDPROC
3499 REM ********************************
3500 DEFPROCmovercursor(tecla)
3501 REM ********************************
3520 NY=RPOS+OFST(2,tecla)
3525 NX=CPOS+OFST(1,tecla)
3540 IF NY > =1 AND NY < =RL THEN RPOS=NY
     ELSE PROCteclanula
3560 IF NX > =1 AND NX < =CL THEN CPOS=NX
     ELSE PROCteclanula
3600 ENDPROC
3999 REM ********************************
4000 DEFPROCactivar—una—celda
4001 REM ********************************
4020 TG=1-BD(RPOS,CPOS)
4025 Y=RPOS+R0:X=CPOS+C0
4040 PRINTTAB(X,Y)D$(TG)
4060 BD(RPOS,CPOS)=TG
4120 PE=?(MPOS+RPOS)
4140 PE=PE+(TG*2-1)*(2^(8-CPOS))
4160 ?(MPOS+RPOS)=PE
4200 PRINTTAB(C0+C9,R0+RPOS)PE
4220 PROCactualizar—texto
4300 ENDPROC
4499 REM ********************************
4500 DEFPROCdefinir—caracter
4501 REM ********************************
4510 REPEAT
4520 PRINTTAB(C0+2,R0+RL+3);
4525 PRINT"                "
4540 PRINTTAB(C0+2,R0+RL+4);
4545 PRINT"                "
4550 PRINTTAB(C0+2,R0+RL+4);
4560 INPUT"NUMERO (224-255)" CARNUM
4580 UNTIL CARNUM > 223 AND CARNUM < 256
4600 CN=CARNUM
4620 PROCvisualizar—caracter
4900 ENDPROC
4999 REM ********************************
5000 DEFPROCcolocar—caracter
5001 REM ********************************
5015 X=CPOS+C0+OF:Y=RPOS+R0
5020 PRINTTAB(X,Y);CHR$(CN)
5040 TX(CPOS,RPOS)=CN
5400 ENDPROC
5999 REM ********************************
6000 DEFPROCvisualizar—caracter
6001 REM ********************************
6020 MPOS=(CN-224)*8+CGEN
6040 FOR R=1 TO CL
6045 PE=?(MPOS+R):P=PE:Z$=""
6060 FOR C=CL TO 1 STEP -1
6065 N=INT(PE/2):Q=PE-2*N
6080 BD(R,C)=Q:PE=N
6100 Z$=D$(Q)+Z$:NEXT C
6120 PRINTTAB(C0+1,R0+R)Z$
6125 PRINTTAB(C0+C9,R0+R)P
6130 NEXT R
6140 ENDPROC
6499 REM ********************************
6500 DEFPROCactualizar—texto
6501 REM ********************************
6520 FOR R=1 TO RL:Y=R0+R
6540 FOR C=1 TO RL:X=C0+C+OF
6560 PRINTTAB(X,Y)CHR$(TX(C,R))
6580 NEXT C,R
6900 ENDPROC
```

Rayos eficaces

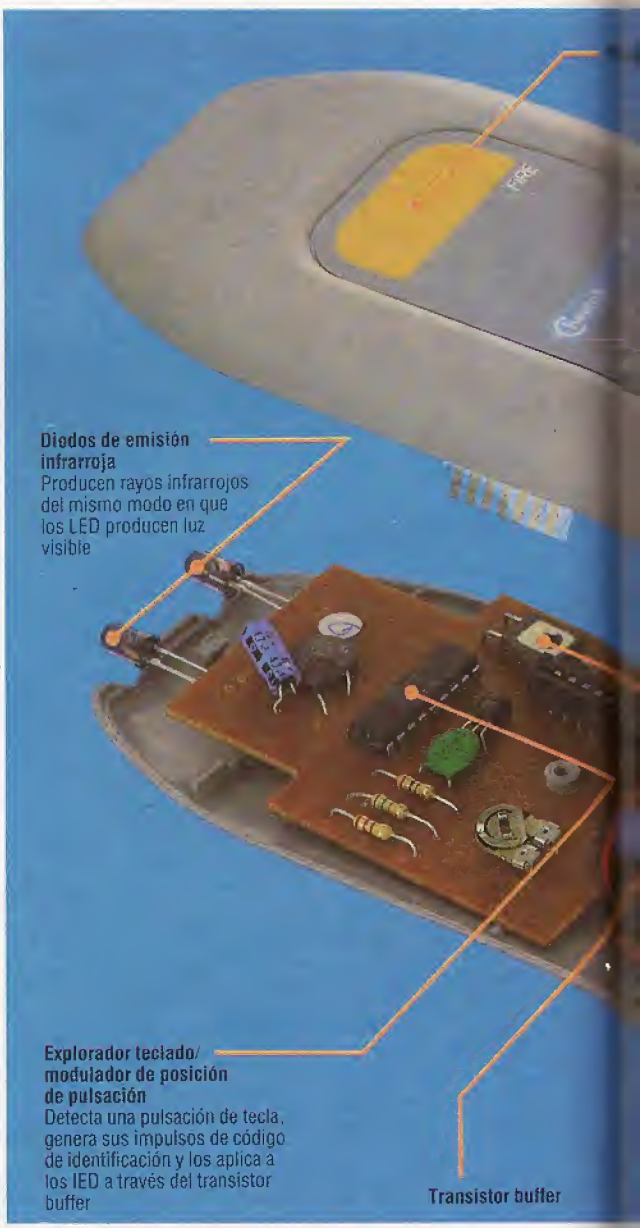
Presentamos la RAT, una palanca a control remoto para el Spectrum que utiliza ondas infrarrojas

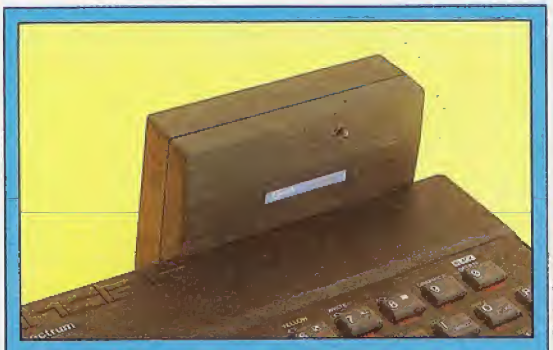
Los buenos aficionados a los juegos recreativos tienden a preocuparse mucho por cualquier cosa que incida significativamente en la velocidad y la calidad de su juego, en especial si también ejerce un impacto considerable en el marcador final. Por este motivo, la forma en que se controla un juego es una cuestión de gran importancia. Con los juegos controlados desde el teclado, las principales cuestiones son la selección de teclas y la facilidad con que las mismas se puedan utilizar. Por consiguiente, los escritores de software deben prestar especial atención a esta clase de detalles. Con los controladores externos, como las palancas de mando, el diseño de hardware tiende a constituir el factor esencial.

La flexibilidad de una palanca de mando (su libertad de movimiento, la rapidez con que reacciona al tacto, la velocidad a la cual responde el juego) debería ser su característica de diseño más significativa. Pero con frecuencia el diseñador se encuentra trabado por las limitaciones impuestas por la naturaleza de la palanca: la longitud del cable de conexión, el tamaño, la forma y la posición del controlador, y la posición del o de los pulsadores de disparo. Este último detalle, por ejemplo, suele favorecer a los jugadores diestros. A pesar de que los fabricantes de palancas de mando han intentado desarrollar diseños que obvien estos inconvenientes, ninguno lo ha conseguido con tanto éxito como la palanca a control remoto por rayos infrarrojos de Cheetah Marketing para el Spectrum.

A su palanca Cheetah la ha denominado RAT. Se dice que el nombre corresponde a las siglas de *remote action transmitter* (transmisor de acción remota), pero más bien parece ser un juego de palabras con la palabra "ratón", que se aplica a los controladores de mano derecha utilizados con el Macintosh de Apple y otras máquinas. La RAT (rata) se parece a un arma láser ligeramente más alargada que la que se ve en la serie de televisión *Star trek* (Viaje a las estrellas). Es larga, plana y gris, con un mando circular para control de color azul, el logotipo de Cheetah y un pulsador de disparo de color naranja brillante. Por el frente de la unidad sobresalen dos transmisores infrarrojos. Cuando uno coge la RAT por primera vez y acciona el pulsador de disparo, casi espera ver saltar de ella rayos de llamaradas azules.

El sistema también incluye su propia interface, que se enchufa en el conector marginal de la parte trasera del Spectrum. Esta caja tiene una puerta para ampliación propia para otros accesorios. En el





Irradiando luz
La radiación infrarroja (que posee mayor longitud de onda que la luz visible pero menor que las ondas de radio) se produce en el transmisor mediante IED (*Infrared Emitting Diodes*: diodos de emisión infrarroja) cuando una corriente eléctrica de un diminuto chip de arseniuro de galio excita las moléculas provocando la liberación de fotones. En el receptor, por el contrario, fluye una corriente eléctrica en el IED cuando la luz infrarroja cae sobre el arseniuro de galio. Por consiguiente, cuando se pulsan los botones de control de la RAT, los dos IED emiten impulsos codificados de infrarrojos en un ancho haz, activando el receptor directamente o después de la reflexión en la habitación

Chris Stevens

Chris Stevens

frente de esta unidad hay un único receptor infrarrojo para comunicación con la RAT.

El paquete viene con una hoja de instrucciones en la que se explica cómo se utiliza la palanca y a qué juegos se puede jugar con ella (todo software que sea compatible con la palanca Kempston). Haciendo un gran alarde de previsión, Cheetah ha incluido rutinas en BASIC y código máquina que permiten incorporar control por RAT en los juegos.

En la hoja de instrucciones se afirma que la RAT se puede utilizar a distancias de hasta 3,5 m apuntando "en la dirección general del ordenador". El movimiento se produce pulsando ligeramente el mando azul de control. En la periferia del mando hay ocho pequeñas protuberancias, y pulsando encima o cerca de ellas se indica la dirección requerida (N, SO, etc., como las direcciones de una brújula). Mientras con una mano se sostiene la RAT y se controla la dirección del movimiento en la pantalla, se puede utilizar la otra para accionar el pulsador de disparo. Debido al diseño de la RAT, el hecho de realizar cada tarea con una u otra mano no supone ninguna diferencia, de modo que la palanca funciona igualmente bien tanto para jugadores diestros como para zurdos. El transmisor requiere una pila PP3, que se coloca en un pequeño espacio en la parte posterior de la unidad directamente debajo del círculo de control azul.

Una vez conectada la caja interface, dotada la unidad transmisora de una pila y cargado en un Spectrum un juego que permita palanca de mando, todo está preparado para ponerse a jugar. Debido a que no hay ninguna señal visible de que el transmisor está funcionando hasta que se ve el movimiento en la pantalla, puede que se sorprenda a sí mismo pegado al ordenador tan cerca como le sucedería con cualquier otra palanca de mando. Existe cierta resistencia a creer que se puede obtener control a una distancia de 3,5 m. Pero cuando uno se da cuenta de que la RAT realmente funciona, entonces desea experimentar para ver desde qué distancia máxima la puede manejar.

De hecho, el transmisor de acción remota funciona sumamente bien a distancias que incluso se acercan a los 4 m. Y no necesariamente hay que tenerlo apuntando hacia la dirección del ordenador. La RAT funciona cuando está orientada directamente hacia el techo, hacia el suelo, de espaldas al usuario o a su lado (si bien le resultará algo difícil ver lo que sucede cuando el transmisor está orientado en ángulos extraños). Evidentemente, la RAT de Cheetah le proporciona al jugador una notable libertad de movimientos. Sin embargo, el mayor inconveniente es que sólo posee ocho posiciones de movimiento: arriba, abajo, derecha, izquierda y puntos intermedios. Sería mucho mejor disponer de un mayor control.

El hecho de que la RAT no posea partes móviles hace que la unidad sea menos susceptible de romperse o desgastarse que las palancas de mando estándares, lo que la hace más durable. De hecho, viene con una garantía de un año. En cuanto a su precio, la RAT Cheetah cuesta apenas un poco más que la mayoría de las otras palancas de mando más la Interface 2. (Por supuesto, si el usuario ya posee esta última, esto no le será de mucho consuelo.) Pero su uso permite muchísima más libertad de movimiento y un control muchísimo mejor que el de la mayoría de las palancas de mando.

PALANCA INFRARROJA CON TRANSMISOR DE ACCIÓN REMOTA CHEETAH

Para: Sinclair ZX Spectrum

Funciona con: Juegos compatibles con la palanca de mando Kempston o la Cheetah

Producida por: Cheetah Marketing, Ltd, 24 Roy St, London EC1, Gran Bretaña

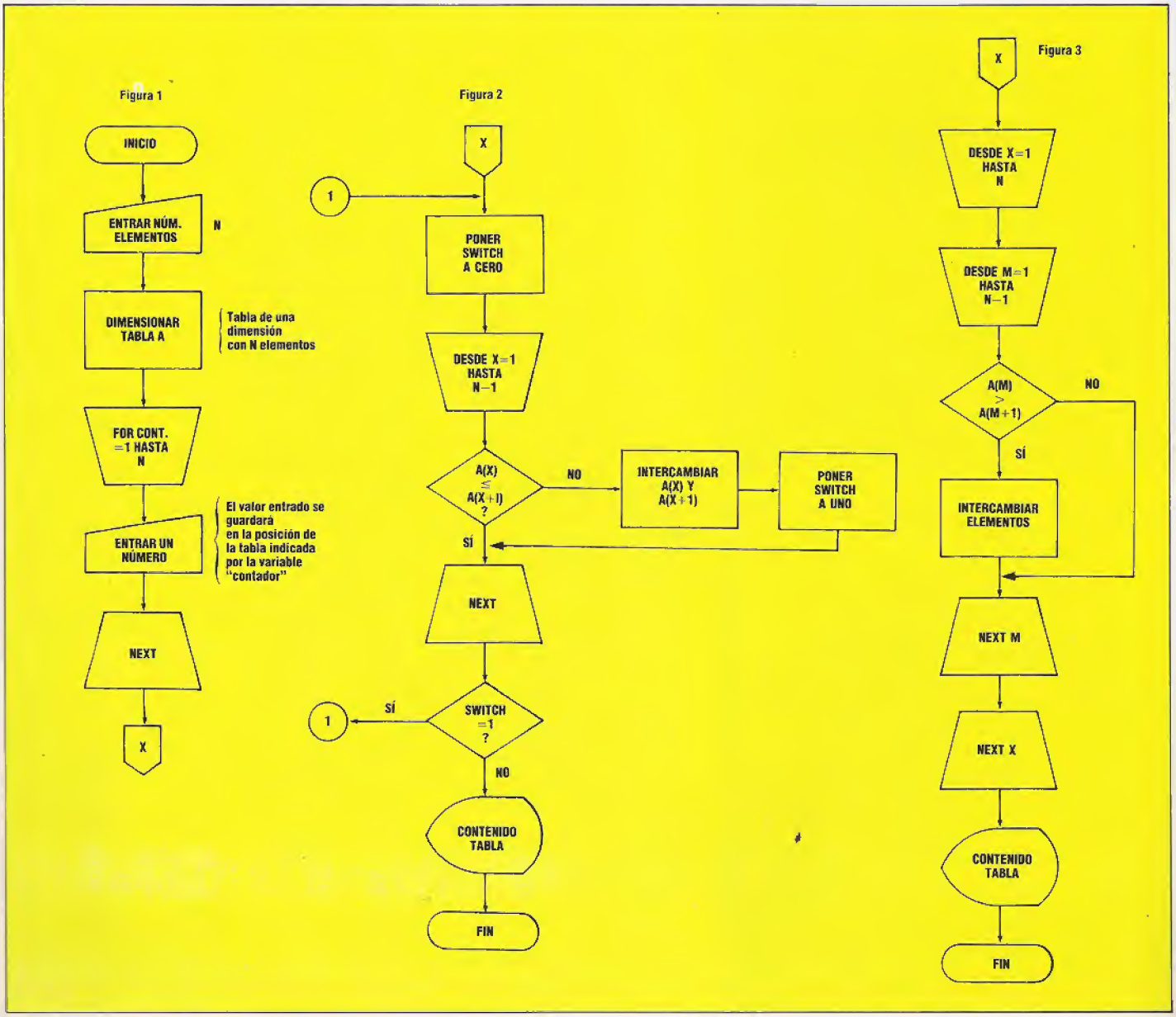
Recepción mezclada
El teclado original del IBM PC Junior, tanto tiempo esperado, se distinguió fundamentalmente por la poca calidad de su aspecto e ingeniería; pero fue, asimismo, el primer microordenador con enlace infrarrojo entre teclado y procesador

Métodos de clasificación

Esta vez nos referiremos a dos eficaces procedimientos, aplicables tanto a valores numéricos como alfanuméricos

Los siguientes diagramas muestran la representación gráfica de dos métodos de clasificación válidos tanto para valores numéricos como alfanuméricos. Son los métodos de "burbuja", llamados así por el hecho de que, tras cada pasada, al menos el último elemento de la lista queda ordenado.

Ambos ejemplos cuentan con una parte común, que es la correspondiente a la carga de la tabla con un número determinado de elementos. Éstos podrían estar ya definidos en sentencias DATA, o bien ser entrados uno por uno desde teclado. Esta parte común es la mostrada en la figura 1, que puede unirse a cualquiera de las dos versiones del citado método, la simple de la figura 2, o la representada en la figura 3 y que requiere del uso de FOR...NEXT anidados.



Variaciones LOGO

Los procedimientos LOGO son más flexibles si el usuario es capaz de pasar parámetros de entrada a dichos procedimientos

En LOGO, una palabra (aquí vamos a tomar como ejemplo TAMAÑO) se puede utilizar de tres formas diferentes. Para distinguir entre las mismas, el LOGO emplea tres notaciones distintas: TAMAÑO, :TAMAÑO (que se lee "puntos tamaño"), y "TAMAÑO ("comillas tamaño"). Como ya hemos visto, si el LOGO se encuentra con la palabra TAMAÑO, con ningún signo de puntuación que la preceda, la toma como el nombre de un procedimiento y llevará a cabo la secuencia de instrucciones especificada en la definición TAMAÑO. :TAMAÑO se utiliza para indicar el *valor* retenido en el nombre de la variable; si el LOGO se encuentra con :TAMAÑO recuperará el valor asociado al nombre. "TAMAÑO se emplea para aludir a nombres de variables y procedimientos, pero indica que nos estamos refiriendo al nombre en sí mismo y no a cualquier valor asociado al mismo. Por consiguiente, "TAMAÑO se puede utilizar para hacer referencia a una variable, mientras que :TAMAÑO hace referencia al valor que se le haya asignado a esa variable. (Observe que se ponen comillas antes de la palabra pero no después de la misma.)

El MIT LOGO no es totalmente coherente en cuanto a la utilización de esta notación. Después de las instrucciones EDIT, ERASE y PO se debe escribir el nombre de un procedimiento sin comillas. Por lo tanto, la sintaxis correcta es EDIT CUADRADO, aun cuando CUADRADO no es una llamada al procedimiento CUADRADO, sino sólo el nombre del mismo, y debería, lógicamente, ir precedido por comillas. El LCSI LOGO es más coherente y requiere el empleo de comillas con estas instrucciones.

Para utilizar cualquiera de los procedimientos que hemos definido hasta ahora simplemente digitamos el nombre del procedimiento, de la misma manera en que utilizaríamos instrucciones del LOGO tales como DRAW o HIDETURTLE. Sin embargo, otras instrucciones (FORWARD, p. ej.) necesitan información extra antes de que puedan utilizarse. La palabra FORWARD sola no tiene ningún significado: se le debe asignar un valor para que el LOGO pueda ejecutar la instrucción. Si incluimos nombres de variables, podemos entrar cualquier valor requerido y, por consiguiente, variar el efecto obtenido cuando llamemos al procedimiento.

Tomemos el procedimiento que definimos en un capítulo anterior para dibujar un cuadrado:

```
TO CUADRADO
  REPEAT 4 [FD 50 RT 90]
END
```

Tal como está, este procedimiento dibujará un cuadrado con lados de 50 unidades de longitud. No obstante, sería muchísimo más útil que el cuadrado se pudiera dibujar en cualquier tamaño escogido; para hacer esto, debemos entrar el valor deseado. Para cambiar CUADRADO de modo que acepte una entrada, utilice el editor para reemplazar el valor fijo de 50 por la variable "LADO y agregue :LADO a la línea del título. Ahora nuestro procedimiento tendrá el siguiente aspecto:

```
TO CUADRADO :LADO
  REPEAT 4 [FD :LADO RT 90]
END
```

Ahora, cuando se llame el procedimiento, será necesario darle un valor a la variable "LADO. Pruebe con CUADRADO 40, CUADRADO 10, etc., para ver cómo varía el tamaño del cuadrado.

Veamos qué es lo que sucede cuando se digita CUADRADO 30. El LOGO primero busca la información de CUADRADO. La línea del título dice que se requiere una entrada y que la misma se va a llamar "LADO. El valor de la línea de entrada (en este caso, 30) se le asigna a la variable "LADO y entonces se obedecen las instrucciones de la definición del procedimiento. La mejor forma de visualizar esto consiste en imaginar que cada nombre de variable alude a una caja que contiene un valor. Cuando el LOGO llega a la línea FORWARD :LADO va a la caja etiquetada "LADO, recupera el valor que allí encuentre y lo utiliza como la entrada para FORWARD. La caja etiqueta "LADO sólo se emplea en el procedimiento que alude a ella. Si existe otro procedimiento que utilice "LADO como el nombre para una entrada, hará uso de una caja diferente. Por consiguiente, se dice que LADO es una variable *local*.

También podemos utilizar entradas con subprocedimientos. El procedimiento CASA que ofrecemos aquí es nuestra solución al problema que planteamos en el capítulo anterior. Está escrito siguiendo las pautas detalladas anteriormente de manera que se puedan entrar valores diferentes:

```
TO CASA :GRANDE
   CUADRADO :GRANDE
   FD :GRANDE RT 30
   TRI :GRANDE
   LT 30 BK :GRANDE
END
TO CUADRADO :TAMAÑO
   REPEAT 4 [FD :TAMAÑO RT 90]
END
TO TRI :LADO
   REPEAT 3 [FD :LADO RT 120]
END
```

Aquí hemos utilizado tres nombres distintos de variable ("GRANDE, "TAMAÑO y "LADO). Podríamos haberles asignado el mismo nombre, dado que las variables son locales de los procedimientos en que se utilizan, pero ello habría dado lugar a confusión.

Para ver cómo funcionan estos procedimientos, veamos lo que sucede si digitamos CASA 30. El LOGO lee la línea de entrada y le asigna el valor 30 a

la variable "GRANDE de CASA. La primera línea de CASA, por consiguiente, ahora es equivalente a CUADRADO 30. A la variable "TAMAÑO de CUADRADO se le asigna, a su vez, el valor 30. Ahora se ejecuta CUADRADO, con FD :TAMAÑO convirtiéndose en FD 30. Un procedimiento similar se sigue luego cuando se llama a TRI.

Ahora intente adaptar los procedimientos para dibujar el tablero de 5×5 de modo que TABLERO tome como entrada el tamaño del cuadrado.

He aquí un procedimiento que dibuja polígonos, con el número de lados dado como entrada:

```
TO POLI :LADOS
  REPEAT :LADOS [FD 50 RT 360 / :LADOS]
END
```

Utilizando este procedimiento con una entrada, POLI 3 dibujará un triángulo, POLI 4 un cuadrado, y así sucesivamente. Sin embargo, en todos los polígonos que dibuje este procedimiento los lados tendrán 50 unidades de longitud. Un procedimiento más general que dibujara polígonos de cualquier tamaño requeriría dos entradas: una para el número de lados y otra para la longitud requerida de los lados. Para hacer esto, todo cuanto se necesita es adaptar el procedimiento POLI para reemplazar 50 por un nombre de variable y agregarle ese nombre a la línea del título:

```
TO POLI :LADOS :TAMAÑO
  REPEAT :LADOS [FD :TAMAÑO RT 360/:LADOS]
END
```

Ahora POLI 5 30 dibujará un pentágono con lados de 30 unidades de largo. Quizá le interesara adaptar su nueva versión del procedimiento para dibujar un tablero de modo que dibuje un tablero de cualquier número de cuadrados (no sólo de cinco por cinco). Ahora deberá haber dos entradas: la cantidad de cuadrados en cada dirección y su tamaño.

Hasta ahora hemos considerado variables que son locales de los procedimientos que las utilizan. Pero se pueden definir variables que estén disponibles para aplicarlas con todos los procedimientos. Se dice que las mismas son variables *globales* y son útiles para comunicar información entre distintos procedimientos. No obstante, su empleo dificulta la tarea de depuración y, por lo tanto, se las debe utilizar con cuidado.

Para asignarles valores a las variables globales se utiliza la instrucción o comando MAKE. MAKE "LADO 3 le asigna 3 al valor de la variable "LADO. MAKE "LADO :LADO+1 incrementa el valor de "LADO en 1. El significado exacto de la notación en este segundo ejemplo es: hallar el valor de la variable "LADO, sumarle uno, luego asignar el resultado a la variable llamada "LADO. En cada caso, MAKE exige dos entradas: el nombre de la variable y el valor a asignarle a ésta.

Para resumir las características de programación que hemos cubierto en este capítulo del curso de LOGO, hemos diseñado algunos procedimientos para dibujar espirales. El procedimiento principal se llama EQESPI. Éste requiere tres entradas: la longitud inicial de la línea a dibujar, el ángulo que se debe describir en cada esquina de la espiral, y un factor de escala en función del cual se debe multiplicar la longitud inicial para producir el efecto en espiral. Se pueden utilizar distintos juegos de entradas para obtener efectos diferentes: nosotros probamos con 70 283 0.95, 70 143 0.95 y 20 243 1.05. Pruebe con otros juegos de números.

NOWRAP es una nueva instrucción o comando. La misma hace que la tortuga no pueda salir de los márgenes de la pantalla: cuando la tortuga llega al límite de la pantalla el procedimiento se detiene con un mensaje de error "fuera de límites". En muchos casos, el efecto de salir por el margen derecho entrando por el izquierdo inmediatamente puede dar resultados interesantes. En este procedimiento frustraría el efecto de espiral, de modo que se utiliza NOWRAP para desactivarlo.

El procedimiento principal EQESPI dibuja repetidamente una línea (cuya longitud está determinada por el factor de escala), luego gira describiendo un ángulo fijo y por último altera el factor de escala. La longitud de la línea dibujada se incrementa o bien disminuye según el factor de escala sea mayor o menor que 1. El número grande después de REPEAT es simplemente para mantener el procedimiento trabajando durante mucho tiempo. Cuando ya haya visto lo suficiente, pulse Control-G (o Break) para detener la ejecución del procedimiento. La mayoría de las variables son locales, con la excepción de "ESCALA. Ésta es global porque "CRECER cambia su valor y este nuevo valor se debe poner a disposición de S.FORWARD. Por consiguiente, se utiliza "ESCALA para la comunicación entre los dos procedimientos.

Procedimiento espiral

```
TO EQESPI :LADO :ANGULO :FACTOR
  PREPARACION
  REPEAT 1000 [S.FORWARD :LADO DERECHO
      :ANGULO CRECER :FACTOR]
END
TO PREPARACION
  DRAW NOWRAP MAKE "ESCALA1
END
TO CRECER :NUM
  MAKE "ESCALA :ESCALA1 :NUM
END
TO S.FORWARD :DIST
  FORWARD :ESCALA1:DIST
END
```

Complementos al LOGO

Las versiones LCSI utilizan la instrucción FENCE en vez de NOWRAP para impedir el movimiento de la tortuga fuera de los límites de la pantalla.
La versión Atari no posee FENCE, de modo que utilice WINDOW. Ésta evita que la tortuga salga de la pantalla, pero no interrumpe el procedimiento cuando llega al límite de la misma.
En el Spectrum, en Espiral sustituya DRAW por CS en el subprocedimiento PREPARACION.

Procedimientos (2)

1) Escriba un procedimiento para dibujar un círculo de radio 50. Modifique el procedimiento de modo que el radio se dé como entrada al procedimiento.
2) Escriba un procedimiento que dibuje una "diana" con cinco círculos concéntricos.

Respuestas a los ejercicios

1) Puzzles tangram

Las formas del hombre sentado, el hombre haciendo reverencia y el gato utilizan el otro lado de la pieza paralelogramo que empleaba el perro del ejemplo de la semana pasada. Para dar vuelta una pieza en LOGO, simplemente cambie todos los giros DERECHA por IZQUIERDA. De modo que en vez de:

```
TO PAR
   REPEAT 2 [FD 35 RT 45 FD 35 RT 135]
END
```

a veces necesitaremos el "otro lado", dado por:

```
TO PAR1
   REPEAT 2 [FD 35 LT 45 FD 35 LT 135]
END
```

Todos los otros procedimientos de piezas son iguales a los expuestos en el capítulo anterior

Hombre corriendo

```
TO CORRIENDO
   MOVER1 TRI1 MOVER2 PAR MOVER3 TRI3 MOVER4 TRI2
   MOVER5 CUADRADO MOVER6 TRI1 MOVER7 TRI2 MOVER8
END

TO MOVER1
   LT 45
END

TO MOVER2
   PU FD 25 RT 135 FD 17.5 LT 45 PD
END

TO MOVER3
   PU FD 75 RT 90 PD
END

TO MOVER4
   PU RT 90 FD 25 RT 90 FD
END

TO MOVER5
   PU FD 50 RT 135 FD 50 LT 135 PD
END

TO MOVER6
   PU RT 135 FD 21 RT 135 FD 25 LT
   90 FD 50 LT 90 FD 25 RT 90 PD
END

TO MOVER7
   PU FD 25 RT 135 FD 71 RT 45 BK 35 PD
END

TO MOVER8
   PU FD 25 LT 90 FD 25 RT 45 FD
   17.5 LT 45 FD 25 RT 135 FD
END
```

Hombre sentado

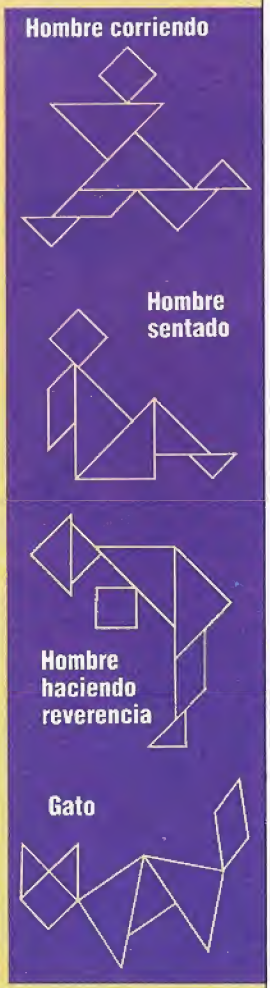
```
TO SENTADO
   MOVER1 TRI1 MOVER2 TRI2 MOVER3 TRI3 MOVER4 TRI1
   PAR1 MOVER5 CUADRADO MOVER6 TRI3 MOVER7
END

TO MOVER1
   LT45
END

TO MOVER2
   PU FD 25 LT 45 FD 17.5 RT 90 PD
END

TO MOVER3
   PU BK 15 LT 90 PD
END

TO MOVER4
   PU FD 50 RT 45 FD 25 RT 90 PD
END

TO MOVER5
   PU FD 25 LT 45 FD 35 LT 45 PD
END

TO MOVER6
   PU BK 50 LT 90 PD
END

TO MOVER7
   PU BK 21 RT 135 BK 50 RT 90 FD 35 LT 90 PD
END
```

Hombre haciendo reverencia

```
TO REVERENCIA
   MOVER1 TRI1 MOVER2 PAR1 MOVER3 TRI3 MOVER4 TRI2
   MOVER5 TRI1 MOVER6 TRI2 MOVER7 CUADRADO MOVER8
END

TO MOVER1
   LT 90
END

TO MOVER2
   PU FD 25 RT 135 FD 70 PD
END

TO MOVER3
   PU LT 45 FD 35 LT 135 BK 50 PD
END

TO MOVER4
   PU RT 90 FD 50 LT 135 PD
END

TO MOVER5
   PU RT 90 FD 50 LT 135 FD 5 RT 90 PD
END

TO MOVER6
   PU RT 90 FD 25 RT 45 FD 7.5 RT 45 BK 35 PD
END

TO MOVER7
   PU FD 35 RT 135 FD 7.5 LT 45 FD 35 RT
   45 PD
END

TO MOVER8
   PU LT 45 FD 36 RT 45 FD 56 LT 135 FD 5 LT
   15 BK 25 PD
END
```

Gato

```
TO GATO
   MOVER1 TRI3 MOVER2 CUADRADO MOVER3 TRI1 MOVER4
   TRI1 MOVER5 TRI3 MOVER6 PAR1 MOVER7 TRI2 MOVER8
END

TO MOVER1
   PU FD 50 RT 90 PD
END

TO MOVER2
   RT 170
END

TO MOVER3
   PU RT 90 FD 25 LT 90 PD
END

TO MOVER4
   RT 180
END

TO MOVER5
   PU RT 90 FD 25 LT 80 FD 50 RT 45 FD 50 RT 90 PD
END

TO MOVER6
   LT 155
END

TO MOVER7
   PU LT 150 FD 35 PD
END

TO MOVER8
   PU FD 35 LT 45 FD 21 RT 135 PD
END
```



Éstas son nuestras sugerencias para las respuestas a los problemas con procedimientos de la p. 1045:

2)

```
TO CASA
   CUADRADO FD 50
   RT 30 TRI LT 30 FD 50
END

TO CUADRADO
   REPEAT 4 [FD 50
   RT 90]
END

TO TRI
   REPEAT 3 [FD 50
   RT 120]
END
```

3)

```
TO CASA
   CUADRADO FD 50
   FD 50 TRI
END

TO CUADRADO
   REPEAT 4 [FD 50
   RT 90]
END

TO TRI
   REPEAT 3 [FD 50
   RT 120]
END
```

4)

```
TO CASA
   CUADRADO FD 50
   FD 50 TRI RT 30 MOVER
   ATRAS
END

TO CUADRADO
   REPEAT 4 [FD 50
   RT 90]
END

TO TRI
   REPEAT 3 [FD 50
   RT 120]
END

TO MOVER
   PU FD 50 RT 30
END

TO ATRAS
   PU FD 50 LT 30 BK 50
END
```

Rapidez y eficacia

En el presente capítulo consideraremos de qué manera se puede aumentar la velocidad de ejecución de los programas

La programación estructurada y un buen diseño de los programas son técnicas que hacen que éstos sean más fáciles de utilizar, pero no mejoran su eficiencia. Para hacer que los programas funcionen más rápidamente y utilicen menos espacio de memoria, a menudo es necesario sacrificar claridad en el diseño de un programa. De modo que debemos tener presente, al "afinar" un trozo de código, que casi todo lo que se haga por hacerlo más veloz invariablemente hará que también resulte más difícil de leer, de entender y de depurar.

La lentitud inherente a los lenguajes interpretados como el BASIC significa que habrá ocasiones en las que los programas se ejecutarán a una velocidad inaceptable y que se los debe acelerar. La forma más eficaz de acelerar un programa en BASIC es compilarlo. Sin embargo, son muy pocos los micros que soportan un auténtico compilador de BASIC. Existen en el mercado compiladores basados en disco y en cassette, pero la mayoría de ellos sólo soportan BASIC con números enteros y, antes de la compilación, puede que requieran un formateado especial de su programa. La compilación es un proceso lento, especialmente durante el desarrollo del programa y sobre todo cuando el sistema está basado en cassette. El compilador ocupará memoria para el usuario y, cuanto más amplias sean sus facilidades, más RAM requerirá del área para programas del usuario. En general, en los micros personales, la compilación sólo es aconsejable para programas totalmente comprobados y depurados.

Los accesos a archivos son, entre todas las causas, las que más retardan los programas. En un programa que lea y escriba frecuentemente en disco o en cinta (un programa de base de datos, p. ej.), las demoras son inevitables. Acceder a un registro en un archivo de acceso directo o al azar contenido en un disco flexible lleva un promedio de un cuarto de segundo, más o menos. El acceso a datos en archivos secuenciales ocupa más tiempo (y varía según la longitud del archivo) y los accesos a cinta son considerablemente más largos. Si estas demoras están causando problemas, tal vez sea posible reducir la cantidad de accesos leyendo más datos por vez y almacenándolos en RAM, y "economizando" actualizaciones de archivos hasta el final de la sesión. A menudo los programas interactivos originan problemas porque el usuario ha de quedarse contemplando la pantalla durante varios segundos. En este caso, una solución parcial consiste en reorganizar el programa de modo que los archivos se lean y se escriban mientras el usuario está ocupado haciendo alguna otra cosa (leyendo una pantalla llena de instrucciones, p. ej.).

Otra causa de lentitud es la aritmética real. Los números reales son los que tienen posiciones decimales (los enteros son números redondos). Debido a la parte decimal, traer un número real de la memoria y efectuar con él una operación aritmética requiere más ciclos de máquina que hacer lo mismo con un entero. En programas con mucha aritmética, siempre que sea posible conviene reemplazar todas las variables reales involucradas por variables enteras (p. ej., SUM se debería reemplazar por SUM%). Se pueden conseguir economías de alrededor del 20 % incluso para programas moderadamente numéricos, y las aplicaciones de "proceso de números" pueden ganar hasta en un 50 %.

Diseñar un algoritmo más rápido es una de las mejores maneras de acelerar un programa. En este curso ya hemos recomendado algunas fuentes de algoritmos. Pruebe con ellas, y esté atento a las que se publican en las revistas de informática. Por otra parte, diseñar algoritmos es una cuestión de creatividad y perspicacia. Los BASIC por lo general poseen una gran riqueza de funciones incorporadas (como INSTR, SGN, LOG, etc.) que son muy rápidas. Esta velocidad es consecuencia de estar escritas en lenguaje máquina y de utilizar los mejores algoritmos disponibles. Con frecuencia vale la pena volver a examinar el manual para ver qué funciones incorporadas se ofrecen antes de codificar una versión propia. Las funciones definidas por el usuario, implementadas mediante la instrucción DEF FN, también operan velozmente. Esta instrucción es de gran utilidad en programas con cálculos repetidos o una secuencia repetida de manipulaciones de series, donde puede sustituir a una llamada a subrutina, que es mucho más lenta.

Escribir subrutinas en lenguaje máquina generalmente hace que funcionen más rápidamente. Ello se debe a que en los programas interpretados deben traducirse sus instrucciones fuente a código máquina a medida que van siendo encontradas durante la ejecución del programa (tampoco esto se

Compiladores para micros

BBC Micro Turbo Compiler Salamander, 17 Norfolk Road, Brighton BN1 3 AA. 0273 771942	Cassette
Commodore 64 DTL-BASIC Compiler Dataview Wordcraft Ltd., Radix House, East Street, Colchester CO1 2XB. 0206 86914	Cassette/disco
Spectrum Softek FP Softek IS, Combined FP and IS, Softek International, 12/13 Henrietta St, London WC2. 01-240 1422	Cassette Cassette Cassette

hace con particular eficacia). Escribir en lenguaje máquina evita este proceso de traducción. Lamentablemente, escribir programas en lenguaje assembly es mucho más difícil que escribir en BASIC, y puede que el costo en tiempo y esfuerzo no compense la eventual economía. No obstante, algunos programas (los que empleen gráficos animados, p. ej.) no funcionarían como se pretende si estuvieran escritos sólo en BASIC.

Existen muchas otras formas de hacer pequeñas economías en velocidad de proceso. Utilice una variable en vez de un número verdadero (p. ej., MAX en vez de 267,5) para un acceso más rápido a los valores, especialmente en bucles. Utilice letras diferentes para comenzar nombres de variables, y seleccione estas letras iniciales de forma uniforme a través del alfabeto. Emplee líneas de sentencias múltiples (si es posible) y cree un intervalo considerable entre los números de línea (como 10). En los bucles FOR...NEXT, si el intérprete lo permite, deje fuera la variable del contador del bucle (p. ej., utilice NEXT en vez de NEXT LOOP). Dentro de un bucle, intente evitar el tener que calcular el mismo valor una y otra vez. En lugar de eso, calcúlelo fuera del bucle e incorpórelo como una variable.

Ahorrando espacio

La aritmética de enteros no sólo ahorra tiempo sino que también ahorra espacio. Cuando almacenar un número real puede gastar cuatro o cinco bytes, sólo se necesitan dos para almacenar un entero. Esto representa un importante ahorro, especialmente cuando están implicadas grandes matrices. Otras mejoras a la velocidad de un programa contribuirán asimismo a ahorrar espacio: utilizar funciones incorporadas o definidas por el usuario economiza código, al igual que escribir en lenguaje assembly y utilizar líneas de sentencias múltiples. La compilación tiende a aumentar el tamaño de los programas más pequeños y sólo ahorra espacio en los grandes.

La eliminación de sentencias REM representa una evidente economía de espacio, y la utilización de textos cortos para los avisos o mensajes también ayuda. La colocación de grandes bloques de texto en archivos almacenados fuera del programa los quita de en medio cuando los mismos no son necesarios (las mayores cargas son los archivos de instrucciones y de "ayuda"). Elimine de cada línea tantos espacios como sea legal, y emplee números de línea y nombres de variables más cortos. Si se necesita dimensionar una matriz pero no se conoce su tamaño exacto, no se limite a aventurar un número redondeado más o menos conveniente; déjela hasta que tenga la información necesaria y después dimensiónela con una variable, como ésta:

```
10 INPUT "Cuantos casos de esta categoria
   hay?";CASOS %
20 DIM MATRIZ %(CASOS %)
```

Esto se conoce como *dimensionamiento dinámico* y es algo que posee el BASIC y que no ofrecen la mayoría de los otros lenguajes; ¡aprovéchelo!

Otra técnica implica aumentar la memoria destinada al BASIC en RAM. Esto se puede hacer mediante la utilización de instrucciones como HIMEM. Lo que suelen hacer estas instrucciones es cambiar el área de RAM disponible para programas en BASIC y variables. La utilización normal de ésta es para almacenar programas en código máquina en un lugar seguro donde no se los pueda machacar por descuido, pero la misma instrucción se puede emplear para acceder a espacio extra del normalmente reservado para la memoria de pantalla. Si lo que está apareciendo en pantalla no tiene importancia, éste es un buen modo de obtener un kilobyte extra de RAM. Si no es posible modificar HIMEM, la memoria de pantalla todavía se puede utilizar sacando y colocando (PEEK y POKE) directamente en las posiciones de memoria reservadas.

Si todo fracasa y el programa simplemente no cabe en el espacio disponible, muchas versiones de BASIC poseen una instrucción CHAIN que permite que un programa le pase el control a otro. Algunos BASIC permiten la utilización de la instrucción COMMON; ésta le pasa al programa siguiente una serie de variables comunes a ambos y sus valores en curso. En algunos micros, CHAIN (en caso de que exista) suele ser una instrucción muy simple que permite pasar al segundo programa todas o ninguna de las variables del primero.

Si los programas están escritos de forma estructurada, las subrutinas individuales deben aceptar ser escritas y verificadas individualmente. Su ejecución también se puede temporizar de manera individual. Escriba un temporizador sencillo como éste:

```
100 REM Usar esta primera seccion para dar valor a
    las variables
105 REM que necesitara la rutina (no olvidar
110 REM dimensionar matrices y llenarlas con
115 REM datos reales si la rutina utiliza alguno).
120 REM Este programa esta en BASIC BBC y TIME
125 REM es una seudovariable que retiene un valor en
130 REM centesimas de segundo, generado por
135 REM el reloj del sistema
200 COMIENZO=TIME
210 GOSUB 2000:REM Aqui se llama a la rutina que
    se va a temporizar
220 FIN=TIME
230 PRINT "La ejecucion
    llevo";(FIN-COMIENZO)/100;"segundos."
240 END
```

Con esta rutina se pueden experimentar diferentes algoritmos y formas de aumentar la velocidad.

Cómo ser rápido

- Sopese concienzudamente los requerimientos de buen estilo con la necesidad de un código veloz.
- Compile cuando pueda hacerlo; si no le es posible, defina funciones y procedimientos.
- Evite los accesos a archivos.
- Evite los números reales absolutos. Inicialice variables y, si su micro lo permite, emplee aritmética de enteros.
- Diseñe sus algoritmos cuidadosamente y aprenda del ejemplo de otros.
- Considere las ventajas y las desventajas del lenguaje máquina. Si bien puede ser más rápido, consume más tiempo en escritura y depuración.
- Condense su código y elimine sus comentarios REM una vez que tenga una versión operativa.

Cambios de domicilio

En este capítulo vamos a examinar con detalle la utilización de los dos registros índice X e Y para el direccionamiento indexado

La intención primitiva del ordenador con programa almacenado fue la de guardar el programa en el mismo lugar (y de la misma forma) que los datos sobre los que iba a operar para que el programa lograra modificarse a sí mismo según se iba ejecutando. El uso más importante no consiste en modificar las instrucciones sino las direcciones donde están los datos sobre los que actúan esas instrucciones. Imagínese el problema de tener que acceder a varios miles de números teniendo que dar instrucciones individuales para cada uno de ellos si cada instrucción sólo pudiera referirse a una dirección imposible de modificar.

Este problema se pudo resolver en parte gracias al concepto de *modificación de dirección*. Por él una misma instrucción puede repetirse tantas veces como se desee y hacer que se refiera a direcciones diferentes de los datos almacenados, mediante el empleo de un valor cambiante en un registro que altere dicha dirección. Este concepto se usa muy a menudo en BASIC. Por ejemplo:

```
FOR I=1 TO N
PRINT TABLA(I)
NEXT I
```

En este caso una misma instrucción, la de PRINT, se refiere a datos diferentes cada vez que se emplea modificando el valor básico (TABLA) mediante el valor indexado (I), que va cambiando reiteradamente.

El principio fundamental del *direccionamiento indexado* es que el contenido del registro índice se agrega a la dirección de base proporcionada junto con la instrucción para obtener así la *dirección efectiva*, es decir, la posición de memoria a la que se accederá en ese instante. Si esta instrucción se produce dentro de un bucle, entonces se puede realizar la modificación por medio del registro índice (incrementos y decrementos por lo general) incluso dentro del mismo bucle, por lo que es fácil acceder a toda una tabla de valores.

El 6809 no sólo tiene dos registros para este menester, el X y el Y, sino otros dos más, el S y el U. Incluso en circunstancias especiales se puede echar mano del contador del programa. Pero el tema del direccionamiento indexado se hace todavía más enredado si se piensa que hay varios tipos de indexación. Con ellos se cubren casi todas las necesidades de programación. De aquí en adelante utilizaremos la indexación de una u otra forma, por lo que tendremos magníficas oportunidades para familiarizarnos con los diferentes modos en que se utiliza.

El direccionamiento indexado se indica colocando X en el campo para operandos, si es que nos servimos del registro X como índice. La forma general de una instrucción indexada sería la siguiente:

```
Opcode   Desplazamiento, Registro índice
LDA   TABLA1,X
STA   TABLA2,Y
```

En muchas ocasiones el desplazamiento es cero, en cuyo caso puede omitirse. Por ejemplo:

```
Opcode   ,Registro índice
LDA   ,X
STA   ,Y
```

Veamos cómo funciona en la práctica. Supongamos que tenemos una tabla de 64 valores de ocho bits almacenados a partir de la posición $3000 y deseamos acceder secuencialmente a los bytes. Podemos definir la dirección base y reservar espacio para la tabla mediante directivas:

```
          ORG     $3000
TABLA     RMB     64
```

Estas instrucciones ponen el contador del programa a $3000, definen el inicio de la tabla en $3000 (TABLA) y reservan los 64 bytes siguientes. Accederemos ahora a los bytes mediante el siguiente paso: el nuevo ORG significa que el código subsiguiente será almacenado en otra parte distinta de la memoria. Cuando se comienza a usar el direccionamiento indexado resulta una medida de precaución recomendable contra la pérdida de control de un bucle, impidiendo así que el programa se destruya a sí mismo:

```
          ORG     $1000
CONT      FCB     0
          LDX     #0
BUCLE     LDA     TABLA,X
```

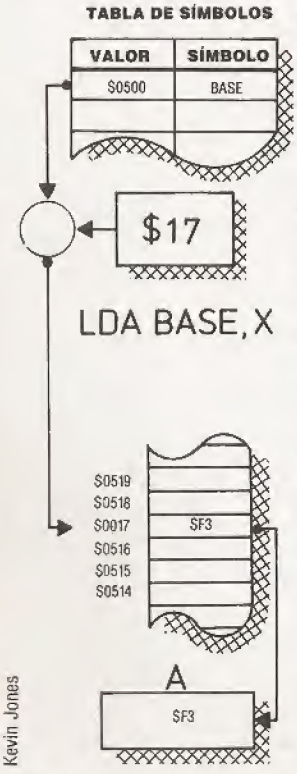
Alteraremos ahora el valor del registro X:

```
          TFR     X,D
          ADDD    #1
          TFR     D,X
```

Sin duda es ésta una manera algo laboriosa de incrementar X, aunque se revela útil cuando se trata de incrementar o decrementar con números mayores de dos. Más adelante estudiaremos otras alternativas a este método. El último fragmento del programa incrementará el contador y hará una comprobación por si se llegó ya al 64 (en cuyo caso se concluye el programa y no hay necesidad de ejecutar el bucle):

```
          INC     CONT
          LDB     CONT
          CMPB    #64
          BLT     BUCLE
```

Enlaces índices
El símbolo BASE, que se inicializa con $0500, es la dirección del primer byte en una tabla de valores. La instrucción en modo de direccionamiento indexado LDA BASE,X toma el valor de BASE y le agrega el contenido del registro X, para obtener la dirección específica del byte cuyo contenido se carga en el acumulador. Si esta instrucción se coloca dentro de un bucle del que X hace de contador, entonces es posible acceder a la tabla entera, byte a byte secuencialmente. Dado que X es un registro de 16 bits, el bucle puede cubrir todo el espacio de la memoria (desde $0000 hasta $FFFF en un sistema de ocho bits, como es el caso del 6809)



Kevin Jones

Existen varias maneras de mejorar la eficacia de este código. Entre las más útiles se encuentra el modo de *autoincremento*. La instrucción

```
        LDA     TABLA,X+
```

hará que el valor contenido en X se incremente automáticamente después de ser utilizado. Si tenemos una tabla de valores de 16 bits tendremos que escribirla así:

```
        LDA     TABLA,X++
```

con lo que el registro X se incrementa en dos. Nuestro programa original resulta ahora mucho más fluido:

```
BUCLE   LDA     TABLA,X+
        INC     CONT
        LDB     CONT
        CMPB    #64
        BLT     BUCLE
```

Una alternativa también útil al método explicado al principio sería recorrer los valores de la tabla en orden inverso, utilizando el modo de autodecremento. Tiene la ventaja de que el valor final del registro X es cero, y dado que el autodecremento de un registro índice activa los flags del registro de código de condición automáticamente, nos será fácil detectar el final del bucle sin necesidad de usar la instrucción CMP. Esto mismo se puede obtener cargando el registro índice con un valor negativo e incrementándolo hasta llegar a cero. Cada vez que se cumple la instrucción de autoincremento, se activan los flags del registro de código de condición para mostrar el resultado del incremento. Si éste es cero, por ejemplo, el que se activa es el flag cero; si se produce un arrastre *(carry)*, se activa el flag de arrastre, etc.

Pero no debemos olvidar la regla general de que es mejor hacer sólo verificaciones de los acumuladores. Y esto porque muchos programas pueden intercalar algún proceso entre la instrucción de incremento (decremento) y la instrucción verificadora, lo cual hace improbable que el registro de código de condición no sufra modificaciones en el código que va de la instrucción a la verificación.

Si decidimos no recorrer la tabla al revés, todavía lo que es posible hacer es que el contador vaya bajando hasta acabar el bucle en cero. Pero hay que estar atentos a que en el modo de autodecremento este último se realiza *antes* del cálculo de la dirección, mientras que en el autoincremento el registro se incrementa *después* de haber sido calculada la dirección. Así, suponiendo que X contiene un 7 y la TABLA comienza en $1000, la instrucción LDA TABLA,X+ cargará el acumulador con el valor contenido en $1000 y seguidamente incrementará X, que pasará del 7 al 8. Por otra parte, LDA TABLA,−X (el signo menos delante del nombre del registro), primero cambiará X del 7 al 8 y después cargará el acumulador con el valor contenido en la dirección $1006.

Nuestro bucle tendrá el siguiente aspecto, si recorremos la tabla al revés y guardamos el contador en el registro B:

```
        LDX     #64
        LDB     #64
BUCLE   LDA     TABLA,−X
        DECB
        BGE     BUCLE
```

El primero de los dos programas que hemos ofrecido como ejemplo muestra un bucle directo que recorre una tabla de valores de ocho bits, donde se cuenta el número de valores negativos. La cuenta llevada por el acumulador B es también empleada como el desplazamiento de un valor fijado en el registro X.

El segundo programa muestra el uso de ambos registros índice conjuntamente, con un desplazamiento cero. Copia la cadena de caracteres colocada en una posición (puede ser en un buffer de entrada, por ejemplo) a otra posición donde será almacenada. La longitud de la cadena es desconocida (contando siempre con que sea menor de 255 bytes), pero concluirá con un carácter RETURN. Una vez almacenado en sus posiciones finales, este carácter RETURN será borrado y se agregará al inicio de la cadena un byte que indique la longitud hasta ese momento desconocida.

Contador de valores negativos

- Se almacena en $3001 una tabla de valores de ocho bits. En $3000 se almacena el número de valores que contiene la tabla que se supone menor de 256. Este programa cuenta el número de valores negativos que contiene la tabla

```
TABLA   EQU     $3000   TABLA contiene el numero de valores
        ORG     $1000
NEGS    FCB     0       NEGS es el lugar donde se guarda el contador de los valores negativos hallados
        LDB     TABLA
        LDX     #TABLA  La dirección de la tabla va a X
BUCLE   LDA     B,X     Toma de la tabla el ultimo valor
        BGE     FINLP   Si el valor es positivo GOTO FINLP
        INC     NEGS    Si no, se cuenta como valor negativo
FINLP   DECB            Recorrido de la tabla
        BGT     BUCLE   Hay mas valores? Si asi es, GOTO BUCLE
        SWI             Si no, se vuelve al sistema operativo
        END
```

Copiador de cadenas de caracteres

- Este programa copia una cadena desde un buffer de entrada en $3000 al siguiente espacio libre para cadenas, cuya dirección se da en $0010

```
CR      EQU     13      Valor ASCII del caracter RETURN
STRING1 EQU     $3000   Dirección del buffer de entrada
STRING2 EQU     $0010   STRING2 retiene la dirección del espacio libre
        ORG     $1000
        LDX     #STRING1 Direccion de la cadena fuente (o sea, $3000 al principio) cargada en X
        LDY     STRING2 Direccion de la cadena de destino (almacenada en $10,$11) cargada en Y
        TFR     Y,U     Paso a U de la direccion del byte de longitud
        LDA     #0      Almacena un cero en el primer
        STA     ,Y+     byte de la cadena de destino
BUCLE   LDA     ,X+     Toma el caracter siguiente del buffer de entrada
        CMPA    #CR     Caracter RETURN?
        BEQ     FINAL   Si no es asi, parar
        STA     ,Y+     Si no es asi, copiarlo (y)
        INC     ,U      Añadir uno a la longitud
        BRA     BUCLE   Caracter siguiente
FINAL   SWI             Volver al sistema operativo
END
```

Apostando al favorito

Este programa de carreras de caballos, creado por Salamander Software para el Oric-1 y el Atmos, es un claro ganador

Ganando etapas
En cualquier etapa del juego los jugadores pueden consultar el programa de carreras (foto superior). Éste proporciona detalles sobre el desarrollo de la prueba y también indica el montante de los premios para carreras específicas. Después de que se han escogido los corredores para una carrera determinada, los jugadores tienen la oportunidad de apostar al caballo de su elección (segunda fotografía). Las apuestas deben oscilar entre 5 y 500 libras. Los caballos se alinean entonces en espera del disparo de salida, y los jugadores se reclinan en sus asientos y aguardan a que empiece la carrera. En la fotografía final vemos a los caballos frenándose tras superar el poste de llegada, con el elegido de nuestro fotógrafo situado en una triste cuarta posición...

Classic racing (Hípica clásica) le ofrece la oportunidad de jugar el papel de un entrenador de caballos de carreras durante una temporada de encuentros. Es un juego para 1-6 jugadores, pero, cuando hay menos de seis personas jugando, el ordenador completa los números, de modo que cada carrera cuenta con seis participantes. El juego le permite elegir la duración de la temporada: una temporada completa abarca 16 reuniones hípicas, cada una de ellas de seis carreras. Quienes tengan menos paciencia seleccionarán una temporada más corta. El objetivo del jugador consiste en ganar la mayor cantidad de dinero posible. Esto se puede conseguir de dos formas: obteniendo el dinero del premio entrenando a uno de los tres primeros caballos de una carrera, o bien haciendo apuestas sobre el resultado. Cada jugador *debe* entrar un corredor para cada carrera, pero no hay nada que le impida apostar a alguno de los corredores de sus contrincantes en caso de que piense que ello le significará una mejor oportunidad de ganar dinero.

Su cuadra consta de 16 caballos y al comienzo de la temporada no se tiene ningún conocimiento acerca de sus méritos. Dado que en las primeras carreras de la temporada los premios son pequeños, ésta es la época ideal para experimentar probando a sus corredores en distintas distancias y con diferentes características de suelo. Esto es simplemente cuestión de ensayo y error: debe observar el rendimiento de un caballo determinado en ciertas condiciones y planificar su estrategia en consecuencia. Ello supone tomar una gran profusión de notas; cada vez que corre un caballo es conveniente apuntar la distancia, el peso transportado, el estado del suelo y el resultado. Es una lástima que Salamander no haya logrado incluir una subrutina para imprimir tales detalles de forma automática, porque ello supondría el ahorro de muchísimo esfuerzo.

Una vez el jugador ha elegido sus seis corredores para la primera reunión hípica, se le darán los nombres de los caballos de sus oponentes y el peso que transportará cada uno. El ordenador hace entonces las apuestas contra cada caballo ganador. Al principio de la temporada esto parecerá hacerse de un modo arbitrario, pero a medida que avanza la temporada, los caballos con malos tiempos partirán con apuestas más bajas. Las apuestas son obligatorias y deben estar entre 5 y 500 libras esterlinas, y las probabilidades ofrecidas pueden ser muy generosas. Dado que un caballo ganador (o colocado) ganará dinero en premio, es ventajoso apostar a alguno de los caballos de los oponentes, con lo cual se tienen dos posibilidades de obtener un beneficio.

También se pueden fraguar "estratagemas" de apuesta, introduciendo a un caballo en carreras para las que obviamente no es adecuado; por ejemplo, un caballo que se desempeñe bien en mil metros en terreno pesado se podría entrar para dos carreras sucesivas de 2 500 m en terreno firme. Es casi seguro que perdería ignominiosamente, después de lo cual se lo podría introducir en una carrera más apropiada con buenas posibilidades. Sin embargo, una vez que se han decidido la distancia y el terreno ideales para un caballo determinado, debe resistirse la tentación de hacerlo correr una carrera tras otra: al igual que en el mundo real de la hípica, los caballos necesitan descansar con frecuencia para obtener su mejor rendimiento.

Cuando se han hecho todas las apuestas, la acción pasa a la carrera propiamente dicha. Los caballos se dirigen a paso cómodo hasta sus posiciones, el juez de salida los llama al orden y después las facilidades de sonido del Oric producen una aproximación bastante fidedigna a los ruidos de los cascos sobre la pista cuando los corredores avanzan hacia la meta. La secuencia de la carrera está lograda de una forma maravillosa: los caballos forcejean por sus posiciones de forma realista y los corredores son tan proclives a tener una reacción impredecible como sus equivalentes de la vida real.

Al final de la carrera se abonan las apuestas ganadoras y el proceso se repite para el resto del programa hasta que finaliza la reunión hípica. Cada encuentro tiene una pista con distintas condiciones de suelo y distancias. Si finalmente decide que uno de sus caballos no está en forma, lo puede retirar de su nómina simplemente no haciéndolo correr en tres encuentros sucesivos. Esto le supone un factor menos de preocupación, pero le vale una penalización de mil libras por cada uno de los encuentros que aún falten.

Hacia el final de la temporada, las carreras van resultando más difíciles de ganar, puesto que para entonces todos los jugadores tienen un concepto mucho más claro acerca del rendimiento de sus caballos y es menos probable que introduzcan corredores en carreras en las que no tienen posibilidades de ganar. Las recompensas son consiguientemente mayores: los tres primeros caballos del Derby, que se corre durante el último encuentro de la temporada, se reparten 90 000 libras en premios.

Classic racing es el programa más atractivo que existe hasta el momento para el Oric y el Atmos. Las secuencias de las carreras son de visión obligatoria, y la planificación total del juego posee un *crescendo* tan apasionante, que hace que el juego siga despertando gran interés aun después de haberlo jugado varias veces.

Classic racing: Para el Oric-1/Atmos de 48 K
Editado por: Salamander Software, 17 Norfolk Road, Brighton BN1 3AA, Gran Bretaña
Autor: Paul Neal
Palancas de mando: No se necesitan
Formato: Cassette

Seres mecánicos

Iniciamos una serie en la que estudiaremos la ciencia de la robótica. En primer lugar, haremos un recuento histórico

Durante siglos muchas personas se han sentido atraídas, de una u otra forma, por la idea y la realización práctica de "hombres mecánicos". Filósofos, ingenieros e inventores se han abocado a la creación de máquinas que imiten el comportamiento humano. A pesar de que en la actualidad los robots se asemejan un poco a los seres humanos y se diseñan para realizar una gama de acciones específicas, los primeros hombres mecánicos se configuraron para que se parecieran en la mayor medida posible al hombre y sugirieran que podían realizar cualquier acción humana.

Al primer robot mecánico, sin embargo, no se le dio forma de hombre. En 1738, el ingeniero francés Jacques de Vaucanson (1709-1782) presentó un pato mecánico ante la Académie Royale des Sciences de París. El pato podía agitar las alas, graznar e ingerir alimentos. Luego, durante el mismo siglo XVIII, un inventor suizo, Pierre Jacquet-Droz (1721-1790), creó un juego de muñecos mecánicos que podían realizar diversas acciones. Uno escribía, otro dibujaba y un tercero tocaba música en un órgano. A fines del siglo XIX ya existía gran número de tales máquinas, todas basadas en mecanismos de relojería.

En Gran Bretaña, durante la época victoriana, se construyeron numerosas figuras de aspecto notablemente natural, y no todas eran de relojería. En 1893 George Moore creó un hombre mecánico cuya capacidad de movimientos se la proporcionaba la energía a vapor; un interesante efecto colateral del mismo consistía en hacer que el hombre mecánico aspirara un cigarro y pareciera exhalar el humo.

Las tecnologías más recientes han estimulado el desarrollo de máquinas más ambiciosas: desde los sencillos hombres mecánicos construidos con piezas del juego Meccano, capaces de caminar, hasta el clásico "Elektro", construido por la empresa norteamericana Westinghouse. El "Elektro" era un hombre mecánico de 2,15 m de altura que podía articular hasta 80 palabras diferentes, contar, caminar, hablar, saludar y distinguir entre distintos colores. Estaba alimentado por no menos de 11 motores eléctricos y pesaba 117 kg. Un "cerebro" compuesto por un total de 82 relés diferentes controlaba esta enorme mole.

Pero cada uno de estos hombres mecánicos tenía sus limitaciones. Ninguno de ellos, a pesar de su evidente valor como fuente de entretenimiento, poseía alguna de las habilidades que uno desearía hallar cuando piensa en el robot ideal. Un hombre mecánico que sepa dibujar no puede ir a hacer la compra por uno, y un hombre mecánico que camine por el cuarto probablemente no llegará ni siquiera hasta la tienda de la esquina sin llevarse por delante algún poste de alumbrado. Cada uno de estos hombres mecánicos era definitivamente una máquina: típicamente, llevaban a cabo una limitada

Colección Kobal

Estrella cinematográfica
El clásico de la ciencia-ficción *Metropolis* (1926), de Fritz Lang, ha influido en el público y en los productores de películas durante décadas, y ha plasmado en La Máquina, primera estrella robot de la historia del cine, las difusas imágenes del progreso

gama de acciones que no exigían la toma de ninguna decisión, y no parecían poseer ningún tipo de inteligencia.

Los robots en la literatura

Pero si los inventores y los ingenieros estaban varados por falta de ideas, los escritores de ciencia-ficción ciertamente no experimentaban estas restricciones creativas. La ciencia-ficción se ha enriquecido con la idea de los robots. De hecho, la propia palabra *robot* procede de una obra literaria. En 1923 el dramaturgo checoslovaco Karel Čapek (1890-1938) escribió una pieza teatral titulada *R.U.R.*; el título correspondía a las siglas de *Rossum's universal robots* (Los robots universales de Rossum). La obra trataba de la invención de un hombre mecánico tan perfecto que era capaz de llevar a cabo trabajos que tradicionalmente sólo podía realizar un ser humano. Finalmente los robots descubrían que no necesitaban a los hombres para nada, lo que dejaba a éstos en una situación poco airosa. En checo la palabra *robota* significa simplemente "trabajo". Por consiguiente, el título de la obra de Čapek se debería haber traducido como "Los trabajadores universales de Rossum"; pero, como quiera que fuera, la palabra "robot" prendió y desde entonces se ha convertido en el vocablo que se aplica a cualquier hombre mecánico que posea habilidades humanas.

Las fantasías de ciencia-ficción acerca de creaturas construidas para asemejarse a los seres humanos se remonta a la famosa novela gótica de Mary Shelley, *Frankenstein* (1818). Si bien no era mecánico, el monstruo creado por Victor Frankenstein estaba conformado a partir de un conjunto de elementos, aun cuando éstos se hubieran obtenido mediante el proceso más bien horripilante de profanar tumbas. Los seres invasores de *La guerra de los mundos* (1898), de H. G. Wells, eran, al menos en parte, mecánicos.

Los escritores del siglo xx, sin embargo, han explorado con enorme detalle un mundo ficticio habitado por robots. La contribución más notable ha sido la de Isaac Asimov, el famoso escritor de ciencia-ficción que comenzó su carrera en 1940 escribiendo relatos sobre robots y sus imaginables problemas operativos. El mundo robótico visionario de Asimov es tan completo que incluso ha formulado las tres "leyes de la robótica". De acuerdo a Asimov, las leyes están contenidas en el *Manual de robótica* (56.ª edición, 2058 d.C.). Evidentemente, el escritor ha concedido un margen de tiempo muy razonable antes de que los robots se conviertan en algo cotidiano.

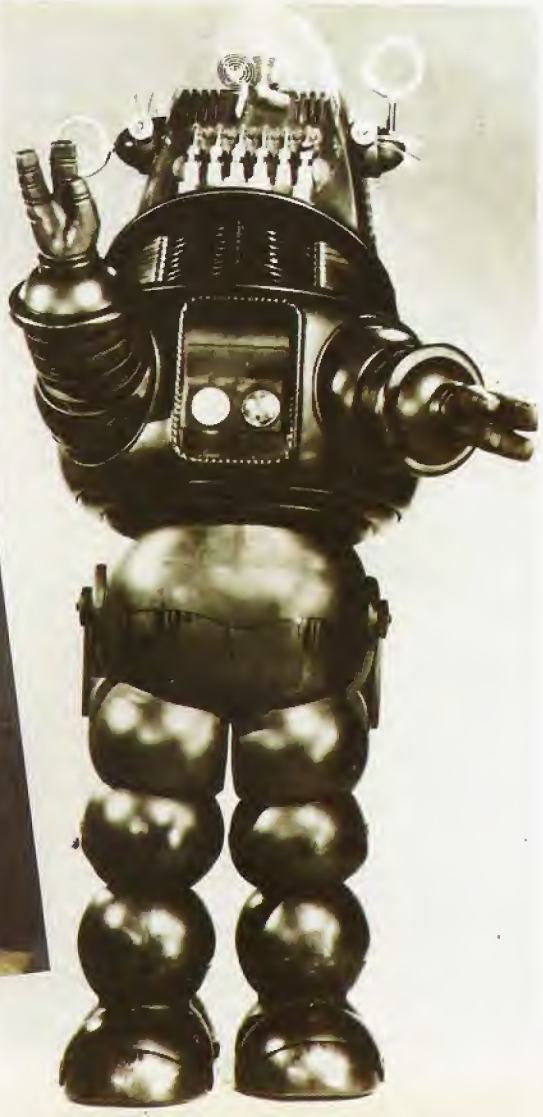
En el cine y la televisión, los robots también han dejado su huella novelesca. La serie de televisión *Dr. Who* está densamente poblada por Daleks y Cybermen, y en las películas de la trilogía de *La guerra de las galaxias* C3P0 y R2D2 están en el mismo nivel de importancia que los protagonistas humanos.

En comparación con estos vuelos de la fantasía, la utilización actual de los robots parece bastante trivial. Los robots industriales de las cadenas de montaje de automóviles reciben en la actualidad la mayor parte de la atención. Se calcula que este año 1985 habrá 25 000 en uso en la industria japonesa, 15 000 en Estados Unidos y 8 000 en la República Federal de Alemania. Se espera que la expansión del mercado europeo para robots industriales siga creciendo sin pausa.

No obstante, muchas personas opinan que los robots industriales son bastante torpes. Una máquina que suelda repetidamente componentes en la carro-

Realidad y fantasía
Los robots más famosos de la televisión europea deben de ser los Daleks. En realidad éstos son armaduras conducidas por personas, controladas desde su propio interior por sus creadores.
Robbie el Robot, de la película *Planeta prohibido*, constituye un buen representante del robot atento y sensible de la leyenda antropomórfica.
Topo *(abajo)*, robot doméstico ya retirado del mercado, fue un intento más o menos serio por introducir la robótica en el hogar

Los Daleks



Robbie el Robot

cería de un coche, o que atomiza pintura sin parar, apenas si representa la imagen novelística del robot. Que el robot ideal llegue a construirse alguna vez ya es una cuestión de conjeturas. Y si tal robot se diseñará a la imagen y semejanza del hombre es igualmente difícil de determinar. Pero examinando más de cerca algunos de los aspectos de la robótica, como haremos en esta serie de capítulos, podremos juzgar por nosotros mismos qué forma podrán tener los robots del futuro.

Glosario de robótica

Los robots, al menos en la ficción, han recibido tal número de denominaciones que tal vez le resulte útil contar con un glosario de los términos más comunes utilizados. Tenga presente, no obstante, que el mero hecho de que a algo se le dé un nombre ¡no implica necesariamente que exista!

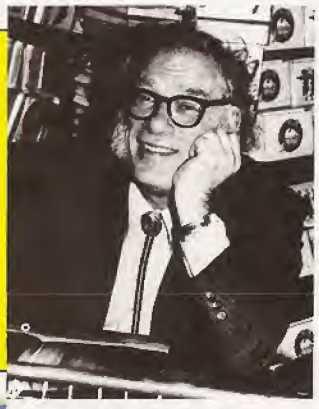
Androide: Robot diseñado para parecerse en todos los aspectos a un ser humano.
Antropomórfico: Literalmente, "de forma humana". Un androide es antropomórfico en todos los sentidos, pero muchos robots están diseñados para ser antropomórficos sólo en algunos aspectos. Por ejemplo, pueden tener un brazo que sea parecido a un brazo humano.
Automatización: Control automático de un proceso de fabricación.
Automatón: Máquina con mecanismos ocultos que normalmente realiza sólo una serie preestablecida de funciones. Los primeros hombres mecánicos eran autómatas. Asimismo, posee un significado más técnico cuando se asocia con *teoría de autómatas*, sistema analítico en virtud del cual se puede estudiar y describir cualquier dispositivo: robots, ordenadores, incluso personas.
Cibernética: Estudio de los sistemas de control y comunicaciones. Ideada por Norbert Weiner en 1947, el reclamo central de la cibernática es que se la puede utilizar para examinar sistemas biológicos como si fueran máquinas.
Cibert: Idea de ficción de un humanoide mecánico.
Ciborg: *Org*anismo *cib*ernético en el cual algunas partes son biológicas y otras mecánicas.
Cibot: Otra idea de ficción; robot con habilidades mentales humanas.
Doppelgänger: Réplica exacta de una determinada persona viviente, aunque por lo general sólo se trata de un espíritu o fantasma.
Droide: Robot bueno que obedece las tres leyes de Asimov.
Efector final: Terminología corriente para la "mano" de un robot.
Homunculi: Pequeños hombres o enanos.
Manipulador: Mano de un robot.
Mecanización: Sustitución de un proceso por un proceso mecánico.
Robot: Máquina capaz de llevar a cabo algunas funciones humanas.
Robótica: Ciencia que estudia los robots.
Trabajadores de cuello metálico: Robots industriales. A los oficinistas humanos a menudo se los llama trabajadores de "cuello blanco", y a los trabajadores manuales se los conoce como trabajadores de "cuello azul". Inevitablemente, se ha comenzado a aludir a los robots como trabajadores de "cuello metálico".

Las tres leyes de Asimov

1. Un robot no ha de herir a ningún ser humano ni, por omisión, ha de permitir que se haga daño.

2. Un robot debe obedecer las órdenes que le son impartidas por los seres humanos, excepto cuando tales órdenes infrinjan la primera ley.

3. Un robot debe proteger su propia existencia en la medida en que dicha protección no infrinja la primera y la segunda ley.



Reglamento para robots
Cuando los robots sean capaces de emprender acciones independientes, las leyes de Asimov podrían constituirse en el fundamento de su conducta. Pero aún son incapaces de identificar a un ser humano, y la regulación jurídica de la interacción robot-hombre todavía es inaplicable

Ford: cadena de montaje del Sierra

Fiat: montaje del eje de torsión

¿Qué es un robot?

Si busca en el glosario, verá que hemos definido a un robot como "una máquina capaz de llevar a cabo algunas funciones humanas", si bien su aspecto no necesariamente parece humano. Obviamente, ésta es una definición muy amplia; se podría aplicar, por ejemplo, a los ordenadores (porque realizan algunas funciones de cálculo). No obstante, en el uso común, un robot tendría algunas cualidades humanas reconocibles. Sería capaz de desplazarse, o quizá incluso de caminar. Podría tener un brazo que se asemejara a un brazo humano. Estaría en condiciones de ver y oír cosas. Hasta podría poseer un alto grado de inteligencia.

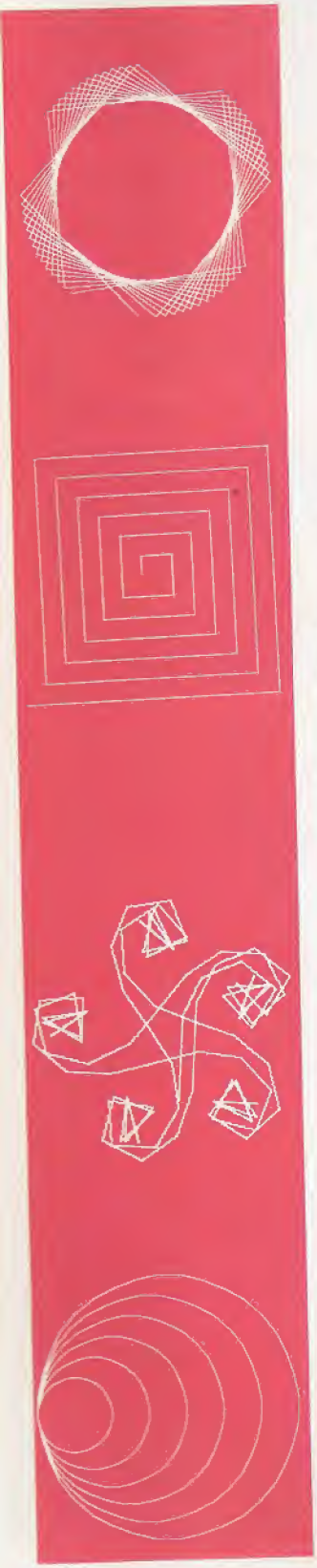
La forma y las habilidades exactas de los robots dependen fundamentalmente de dos factores: lo que deseamos que hagan y lo que podemos conseguir que hagan. Por ejemplo, un robot industrial utilizado para soldar puede no ser capaz de desplazarse, no porque nosotros no podamos hacer que se desplace, sino porque deseamos que permanezca en un sitio y se dedique a cumplir con la misión asignada. Del mismo modo, un robot doméstico podría ser capaz de preparar una taza de té, pero quizá no pudiera subir las escaleras para llevársela a usted hasta su dormitorio.

El término "robot" se ha convertido en la palabra genérica para todas las máquinas parecidas a los humanos, y los límites acerca de lo que son y de lo que pueden hacer conciernen a quienes los diseñan y los construyen.

Montaje robot
Durante algún tiempo más, los robots se utilizarán básicamente en las cadenas de producción. La economía de la producción masiva hace de ellos unos trabajadores ideales para cadenas de montaje, tal como muestran claramente las factorías de Fiat y Ford. La especialización suele exigir que estos robots se reduzcan a uno o dos brazos equipados con pinzas, llave de tuercas y aparatos para soldar

Actuación repetida

El LOGO utiliza la técnica matemática de la recursión para crear complejos diseños mediante sencillas instrucciones



Uno de los primeros programas que definimos en el curso era un procedimiento para dibujar un cuadrado. La definición le indicaba a la tortuga que avanzara una cierta distancia, que girara 90° a la derecha y que repitiera esos dos pasos tres veces más. Ésta es otra forma de dibujar un cuadrado:

```
TO CUADRADO
  FD 50
  RT 90
  CUADRADO
END
```

Al ejecutar esto, la tortuga dibujará un cuadrado y luego seguirá desplazándose a lo largo del perímetro del cuadrado hasta que se interrumpa con Control-G o BREAK. Lo más destacable de este nuevo procedimiento CUADRADO es que se llama a sí mismo; en otras palabras, que es *recursivo*.

Cuando se ejecuta este procedimiento, el LOGO va a buscar la definición de CUADRADO y empieza a obedecer las instrucciones. La tortuga se mueve FORWARD 50 y luego gira RIGHT 90. La siguiente instrucción es CUADRADO, de modo que el LOGO carga a memoria la definición de CUADRADO y empieza a ejecutarla. Esto se producirá *ad infinitum* si no se interrumpe el programa.

También se pueden utilizar llamadas recursivas en procedimientos que exijan inputs:

```
TO POLI :LADO :ANGULO
  FD :LADO
  RT :ANGULO
  POLI :LADO :ANGULO
END
```

Este procedimiento puede producir todos los polígonos que hemos definido hasta ahora en el curso (véase p. 1025), así como muchos de los que no nos hemos ocupado (pruebe a utilizar este procedimiento con un valor de ángulo de 89). También es posible cambiar el valor de la entrada en la llamada recursiva. De esta manera:

```
TO POLIESPI :LADO :ANGULO
  FD :LADO
  RT :ANGULO
  POLIESPI (:LADO+5) :ANGULO
END
```

La única diferencia entre este procedimiento y POLI es que se le suma 5 al valor de LADO cada vez que se lo llama. De modo que si se empieza con POLIESPI 10 90, la primera llamada dibujará una línea de longitud 10, la segunda será de 15, luego 20, etc. El resultado será una espiral. Quizá le agrade experimentar con entradas diferentes: 10 90, 10 95, 10 120, 10 117, 10 144 y 10 142 son buenos puntos de partida. También puede tratar de modificar el procedimiento: una posibilidad es cambiar la suma por resta o multiplicación.

El siguiente es un procedimiento similar que incrementa el ángulo en vez del valor del lado:

```
TO ENESPI :LADO :ANGULO :INC
  FD :LADO
  RT :ANGULO
  ENESPI :LADO (:ANGULO + :INC):INC
END
```

Pruebe diversas inputs: 5 0 7, 10 40 30, 15 2 20, 5 30 20 servirán para empezar. ¿Por qué algunas formas se cierran y otras no? ¿Puede descubrir alguna regla?

La simple repetición de un trozo de código se denomina *iteración*. Con esta finalidad el LOGO utiliza REPEAT (repetir), mientras que otros lenguajes emplean una variedad de construcciones, como FOR...NEXT, REPEAT...UNTIL y WHILE...WEND. Sin embargo, el LOGO se basa mucho más en la recursión que en la iteración. Si ha programado en otros lenguajes puede que tenga cierta dificultad para evitar la utilización de la iteración, pero los gráficos tortuga son ideales para experimentar con llamadas recursivas.

Reglas para interrupción

Todos los procedimientos recursivos que hemos analizado hasta ahora continúan repitiéndose indefinidamente. Es evidente que necesitamos una manera de hacer que un procedimiento se detenga en algún punto. Tomando como ejemplo el procedimiento CUADRADO, un posible lugar para interrumpirlo sería después de que hubiera dibujado un cuadrado completo y el encabezamiento de la tortuga estuviera otra vez en 0. Esto se puede hacer agregándole al procedimiento un "método de parada":

```
TO CUADRADO :LADO
  FD :LADO
  RT 90
  IF ENCABEZAMIENTO=0 THEN STOP
  CUADRADO :LADO
END
```

Las nuevas primitivas son STOP e IF. La primera de estas instrucciones hace que la ejecución de un procedimiento se interrumpa y el control retorne al procedimiento que lo llamó. Una sentencia IF es la forma del LOGO de tomar decisiones. IF va seguido de una condición, y THEN por una acción que se efectúa si la condición se cumple.

Veamos una versión de POLIESPI con un método de parada y consideremos exactamente qué sucede cuando se ejecuta:

```
TO POLIESPI :LONGITUD
  IF :LONGITUD > 15 THEN STOP
  FD :LONGITUD
  RT 90
  POLIESPI (:LONGITUD + 5)
END
```

Veamos qué es lo que sucede cuando ejecutamos POLIESPI 10. Se llama al procedimiento POLIESPI y se define una variable local inicializándola a 10. Puesto que este valor no es mayor que 15, el LOGO procede a ejecutar el movimiento FD 10 RT 90 y después efectúa una nueva llamada a POLIESPI, pero esta vez con un valor de entrada de 15. Esto hace que se vuelva a llamar a una copia del procedimiento. Dado que LONGITUD no es mayor que 15, la tortuga se mueve FD 15 RT 90 y se realiza otra llamada a POLIESPI. Pero esta vez la variable local se ha incrementado a 20, de modo que el procedimiento se interrumpe y el control retorna al procedimiento que lo llamó (POLIESPI 15). Este procedimiento, a su vez, ha llegado a su línea final y le devuelve el control a su procedimiento de llamada. Éste también se interrumpe, en cuyo momento el programa ha llegado a su final.

Hemos visto cómo en LOGO la recursión implica procedimientos que llaman a copias de sí mismos. Es importante tener presente que las llamadas recursivas son copias que existen del procedimiento original, trabajando como si fueran completamente independientes del mismo. Al acabar, dicho procedimiento siempre le devuelve el control al procedimiento que lo llamó. Para ilustrar más claramente el proceso de retornar desde llamadas a procedimientos, podemos reescribir POLIESPI de la siguiente manera:

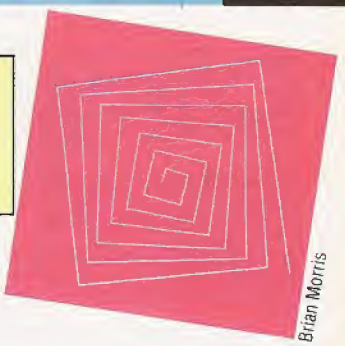
```
TO POLIESPI :LONGITUD
  IF :LONGITUD > 15 THEN STOP
  POLIESPI (:LONGITUD + 15)
  FD :LONGITUD
  RT 90
END
```

Al ejecutarlo verá que el programa hace su dibujo "hacia atrás": las líneas se dibujan en espiral hacia adentro en vez de hacia afuera. (Esto se apreciará más claramente si se utiliza en la sentencia de condición un valor mayor: por ejemplo, utilizando 50 en lugar de 15.) Lo que es significativo aquí es que el LOGO dibuja cada línea cuando se devuelve el control desde las llamadas a procedimientos. En nuestros ejemplos anteriores se dibujaba una línea y el control pasaba entonces a otro procedimiento. Pero aquí todos los procedimientos son llamados antes de empezar ningún dibujo, y el último valor creado de LONGITUD es el que se utiliza primero.

Por último, debemos observar que la recursión es una técnica que ocupa muchísima memoria. Los procedimientos en los cuales la llamada recursiva está en la última línea son los que se implementan más eficazmente, dado que no ocupan ninguna memoria adicional independientemente de cuántas veces sean llamados. Si se puede escribir un procedimiento de modo que sea "recursivo al final", por lo general siempre vale la pena hacerlo.

Procedimientos (3)

Escribir un procedimiento recursivo para dibujar una torre de cuadrados uno encima del otro, reduciendo cada vez la longitud del lado a la mitad

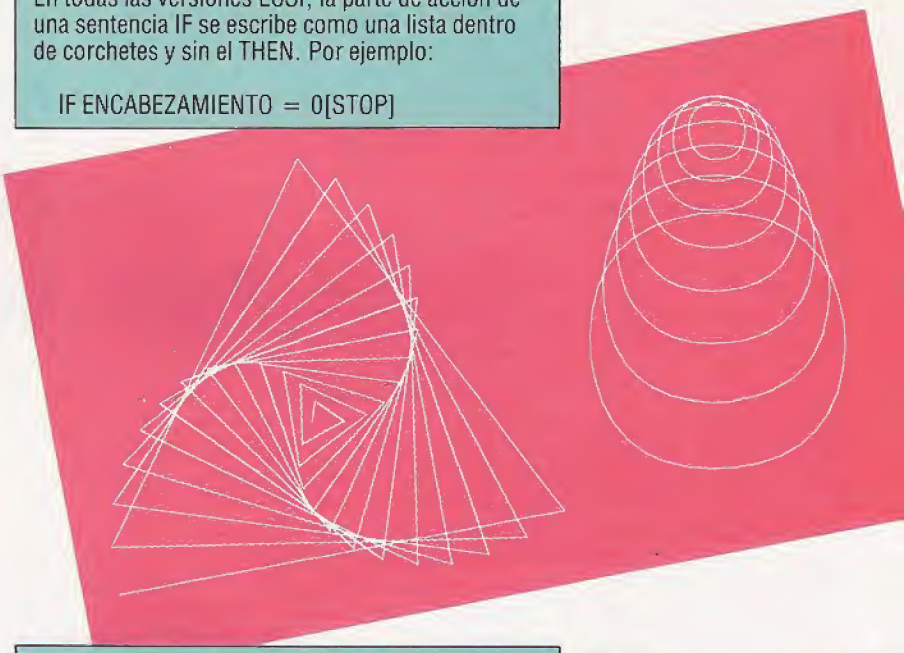


Brian Morris

Complementos al LOGO

En todas las versiones LCSI, la parte de acción de una sentencia IF se escribe como una lista dentro de corchetes y sin el THEN. Por ejemplo:

```
IF ENCABEZAMIENTO = 0[STOP]
```



Respuestas a ejercicios

Un procedimiento para dibujar un círculo, dando el radio como entrada, con la posición actual como un punto de la circunferencia:

```
  TO CIRCUITO :RADIO
    REPEAT 36 [FD (2*:PI*:RADIO/36) RT 10]
  END
MAKE "PI(3.14159)
```

Si adaptamos el procedimiento para que el centro del círculo esté en la posición actual, tendremos:

```
  TO C.CIRCULO :RADIO
    PU LT 90 FD :RADIO RT 90 PD
    CIRCULO :RADIO
    PU LT 90 BK :RADIO RT 90 PD
  END
```

BLANCO utiliza C.CIRCULO para dibujar 5 círculos concéntricos:

```
  TO BLANCO
    C.CIRCULO 10 C.CIRCULO 20 C.CIRCULO 30
    C.CIRCULO 40 C.CIRCULO 50
END
```

Comprobación de rutinas

En este capítulo final analizamos los métodos que se pueden utilizar para probar un programa ya terminado

Una de las grandes ventajas de programar en un lenguaje interpretado como el BASIC es que el código se puede verificar a medida que se va escribiendo. El programador puede en cualquier momento entrar RUN y ver lo que sucede. En la mayoría de las máquinas, es muy fácil interrumpir la ejecución de un programa, imprimir (PRINT) los valores de variables clave, modificar estos valores y después CONTinuar. Todo esto significa que se pueden detectar y corregir la mayoría de las equivocaciones. Pero esta especie de depuración *ad hoc* no es un sustituto de la comprobación formal, que se debe realizar cuando el programa se haya completado.

La prueba definitiva pretende asegurar que un programa hará exactamente lo que está destinado a hacer. Para cualquier conjunto posible de datos de entrada correctos debe producir la salida correcta, y para toda entrada ilegal debe emprender las acciones apropiadas. Una forma sencilla de verificar un programa podría ser la de darle una muestra de todas las entradas legales posibles y comprobar después que los resultados son los esperados. Sin embargo, para la mayoría de los programas esto será inviable. Imaginemos un programa que tome dos enteros, los sume e imprima el resultado. Siguiendo este método, habrá de ser ejecutado ¡para todos los valores enteros posibles! Y ésta es sólo una parte del problema, puesto que también habrá de probarse con todos los valores *ilegales*.

Otra posibilidad podría ser examinar todos los "caminos" a través del programa. Se puede descubrir un camino determinado siguiendo una ruta a través de un diagrama de flujo desde el principio hasta el final. Cada bifurcación que aparezca en el recorrido abre caminos alternativos y cada bucle va agregando más. La figura 1 muestra un programa que es un bucle que contiene unas cuantas sentencias IF...THEN. Dentro del cuerpo del bucle hay cuatro caminos y el bucle se ejecuta 10 veces. Esto significa que la cantidad de rutas exclusivas desde "comienzo" a "final" es de 1 398 100: una cifra asombrosa para lo que probablemente no consista en nada más que en una docena de líneas de código.

Por consiguiente, si la comprobación exhaustiva de datos no funciona y la comprobación a fondo de la lógica tampoco, ¿qué es lo que sí funciona? La respuesta, sorprendente, es nada. No existe ninguna forma de probar completamente un programa complejo en un tiempo razonable. En parte por esta razón, la comprobación sigue la ley de que la cantidad de errores hallados por unidad de esfuerzo disminuye con cada unidad extra. En consecuencia, el momento adecuado de dejar de buscar errores es cuando el esfuerzo de hacerlo importa más que el costo de los fallos aún no detectados.

Cuatro simples condiciones

Incluso una construcción tan simple como este bucle no se puede verificar exhaustivamente debido a la multiplicidad de posibles condiciones de entrada: a través del bucle hay más de un millón de rutas únicas

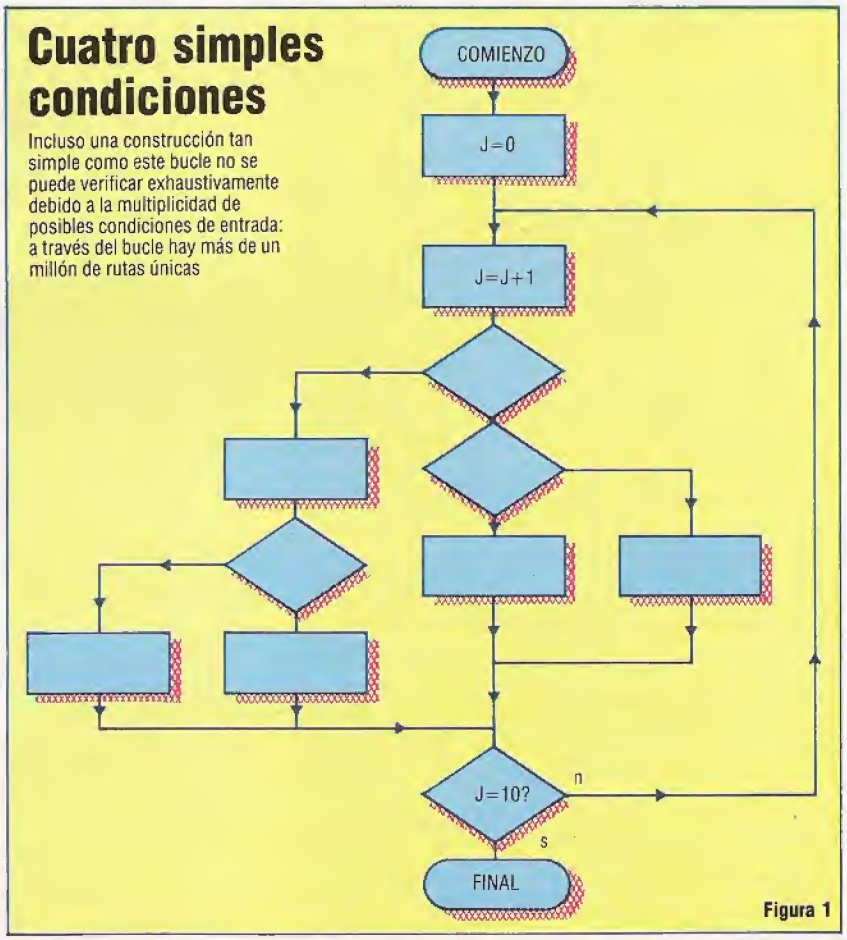


Figura 1

Sin embargo, a pesar de estos inconvenientes, vale la pena desarrollar algún método de comprobación. Una presunción razonable es que si una máquina opera correctamente con un dato de un tipo determinado, operará correctamente con todos los datos del mismo tipo. De modo, pues, que si una subrutina funciona para un entero positivo dentro de su escala, deberá funcionar para todos los enteros positivos de esa escala. Esto nos lleva a una clase de comprobación conocida como *comprobación de clases de equivalencia*. La idea consiste en desarrollar un conjunto de casos de prueba cada uno de los cuales sea representativo de una clase de casos que se comportarán de la misma manera. Por lo tanto, si un trozo de código comprueba que una entrada está comprendida entre 1 y 100, deberemos verificar las entradas que sean menores que el menor valor esperado, mayores que el mayor valor esperado, y que se encuentren dentro de la escala esperada (valor < 1, valor > 100, y 1= < valor= < 100).

El examen de cada uno de los caminos lógicos también se puede simplificar llamando a cada rutina por todos sus puntos de entrada (si bien idealmente debería haber sólo uno) y, dentro de cada rutina, cubriendo cada una de las posibles salidas de todas las decisiones. En la figura 2 tenemos una rutina para ajustar los puntos de premio de un juego. Toma los parámetros de entrada PREMIO,

Sólo probando
Un juego completo de datos de prueba calculados de antemano para el ejemplo ilustrado en los diagramas de flujo podría tener el siguiente aspecto:

	ENTRADA		SALIDA
NIVEL	ACIERTOS	PREMIO	PREMIO
6	10	200	1300
4	10	550	2300
7	10	550	3950
4	10	200	800
7	10	200	1400
1	20	2500	2600
1	20	550	550
6	5	200	300
6	50	200	300
4	5	2500	2600
7	50	2500	2600
4	50	550	550
7	5	550	550

NIVEL y ACIERTOS y devuelve un valor (posiblemente nuevo) para PREMIO. Se podría escribir así:

```
6030 IF NIVEL > 2 AND ACIERTOS=10 THEN
     PREMIO=PREMIO*NIVEL
6040 IF NIVEL=6 OR PREMIO > 2000 THEN
     PREMIO=PREMIO+100
```

Para cubrir la salida de cada expresión condicional necesitamos considerar las entradas a cada una que producirían una salida de "sí" o "no". En ambas decisiones estamos observando los efectos de dos variables combinadas mediante un operador lógico (AND y OR). Ello significa que tenemos que tomar en consideración los valores combinados de las variables y no sus valores individuales. Para ver por qué, consideremos lo que sucedería si comprobáramos valores para NIVEL de 4 y 1 y para ACIERTOS de 10, 5 y 20 en la primera decisión. Cuando NIVEL=4, se comprueban los tres valores de ACIERTOS, pero cuando NIVEL=1 no son comprobados. Esto es un caso en el cual una parte de una decisión "enmascara" a otra. Para comprobar cada parte por separado, es mejor simplificar las decisiones compuestas.

Observando la figura 3, podemos ver que con cuatro decisiones binarias hay 2^4 (=16) posibles resultados y debemos cubrirlos todos. Un punto de partida es hacer la lista de las condiciones para un resultado sí o no para cada decisión, como ésta:

	1	2	3	4
sí	NIVEL>2	ACIERTOS=10	NIVEL=6	PREMIO>20000
no	NIVEL=2 NIVEL<2	ACIERTOS<10 ACIERTOS>10	NIVEL<6 NIVEL>6	PREMIO=2000 PREMIO<2000

Las mismas se pueden utilizar para deducir los valores para datos de prueba representativos. Por ejemplo, para el camino adfi (véase fig. 3), NIVEL debe ser mayor que 2 e igual a 6, ACIERTOS no debe ser igual a 10 y PREMIO puede tener cualquier valor (porque no está implicado). Los valores NIVEL=6, ACIERTOS=20 y PREMIO=150 emplearían este camino (al igual que muchos otros, por supuesto). La ruta abehj se podría probar con NIVEL=4, ACIERTOS=10 y PREMIO=600 (no olvide que estamos hablando del valor de *entrada* de PREMIO que posteriormente puede ser multiplicado por NIVEL).

Es igualmente importante el hecho de que los resultados que debe producir cada uno de los conjuntos de datos se deben calcular antes de la ejecución de prueba, de modo que se puedan comparar los resultados. Los datos de entrada solos, únicamente probarán si el programa funciona. Para comprobar que esté efectuando lo que debe hacer, la salida se debe calcular a mano previamente. Podemos ver un conjunto completo de casos de prueba para este ejemplo (izquierda).

Ya en posesión de un método para "ejercitar" nuestro software, ahora necesitamos una forma de abordar un programa más largo de modo que la complejidad no resulte abrumadora. Es aquí donde se percibe otro de los beneficios de la programación estructurada. Los programas escritos como un conjunto de módulos independientes dispuestos según una jerarquía nos permiten probar cada módulo de forma individual. Dado que los módulos están dispuestos de esta manera, podemos empezar por el módulo superior e ir trabajando hacia abajo, comprobando cada módulo individual sólo cuando ya se han verificado todos los que hay por encima del mismo, y podemos utilizar módulos ya verificados para proporcionar datos a los módulos inferiores de la estructura.

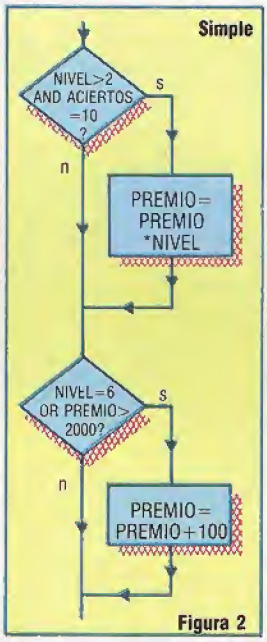


Figura 2

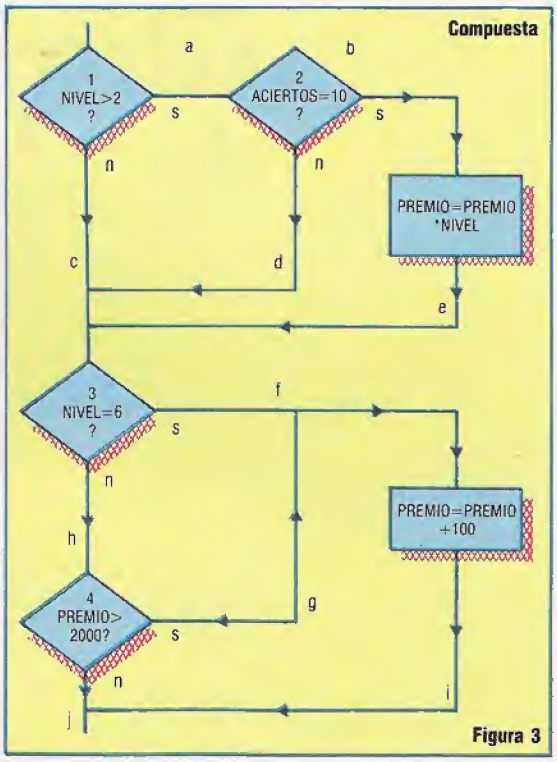


Figura 3

Enmascarar decisiones
La simplificación de decisiones y el etiquetado de los enlaces ayudan a la verificación

El módulo que se esté probando tendrá por encima (a menos que sea el primero), un módulo *conductor* totalmente probado. Los módulos por debajo de él están todavía sin comprobar y por consiguiente no son fiables, de manera que se simulan mediante cortos trozos de código que simplemente devuelven los datos de prueba apropiados cuando son llamados por el módulo que se está verificando. Para usar esta técnica se suele emplear un esqueleto de código en el cual se pueden colocar las rutinas modulares para su comprobación. La figura 4 ilustra este principio. Los módulos 1, 2 y 3 ya se han comprobado, mientras que los módulos 5, 6 y 7 son simulados.

Debemos hacer hincapié en un último punto. La comprobación es una parte importante del ciclo vital del programa y, como tal, merece estar bien documentada.

Comprobación de arriba abajo (top-down)
La comprobación se simplifica muchísimo mediante el enfoque de arriba abajo, porque cada módulo se puede verificar después de ser escrito, tanto de forma aislada como en asociación con otros módulos ya comprobados. El comportamiento de los módulos aún no escritos se puede simular escribiendo pequeñas rutinas ficticias que generen artificialmente ejemplos de la salida prevista del módulo

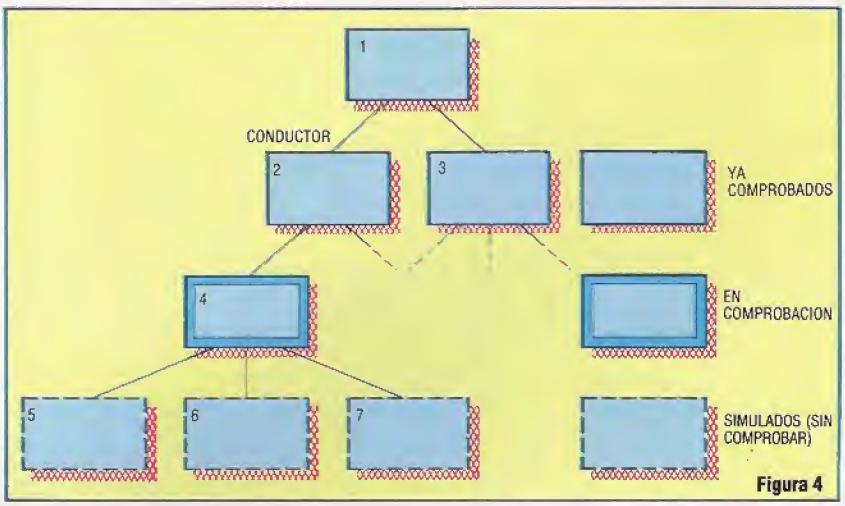
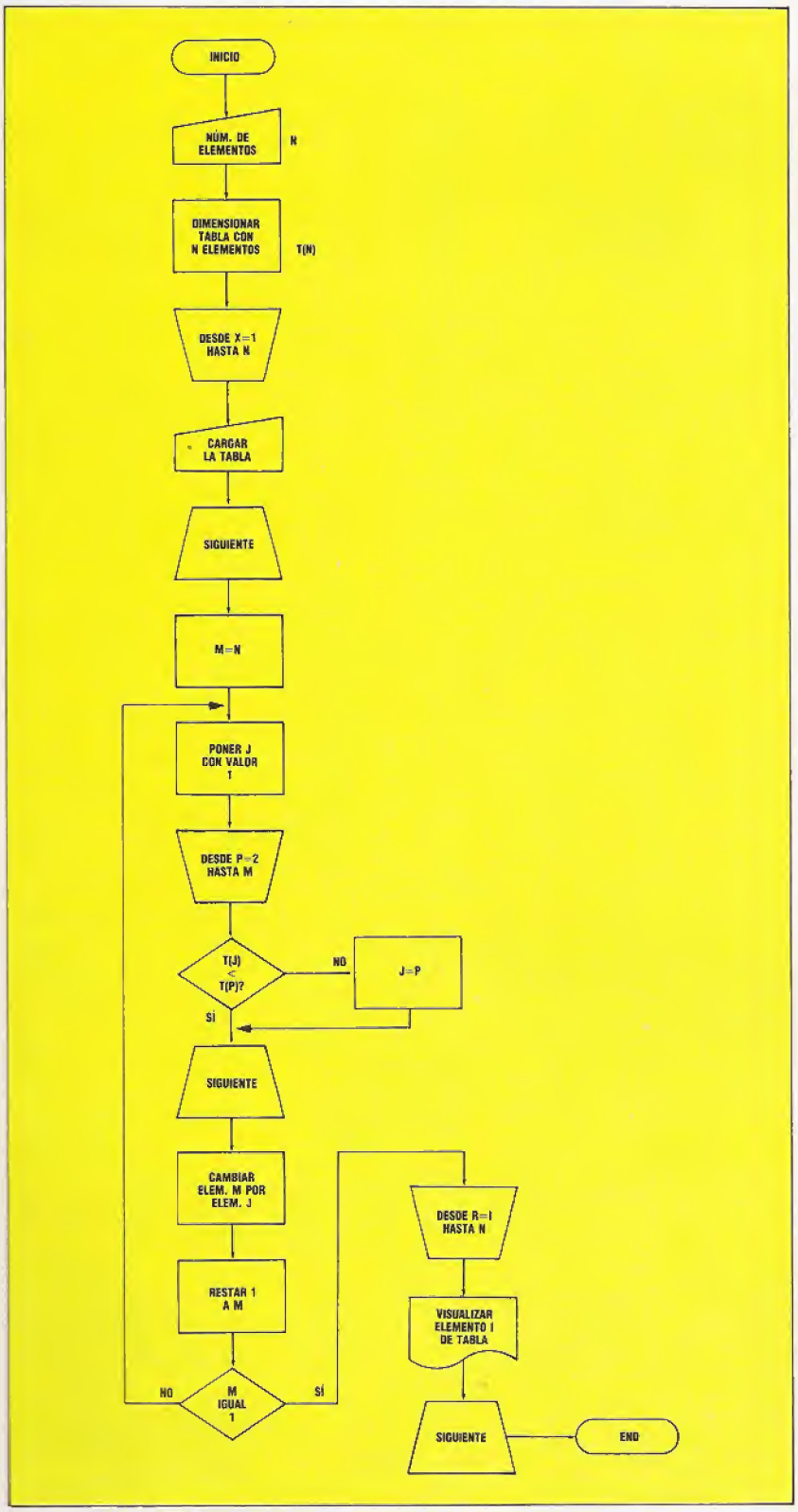


Figura 4

Liz Dixon

Otro método de clasificación



He aquí otro eficaz procedimiento de clasificación, en que ésta se efectúa por el máximo

En el capítulo anterior examinamos la representación gráfica de dos métodos de clasificación válidos tanto para valores numéricos como alfanuméricos, los llamados métodos de "burbuja".

El empleo que presentamos en esta ocasión está realizado mediante la llamada técnica de clasificación por el máximo. Consiste en tomar una serie de datos, que, uno detrás de otro, se van disponiendo en el lugar correspondiente, siempre según su valor dentro de los elementos que componen la lista. Al comienzo de la misma quedan aquellos que tengan un valor más bajo; siguen los que tengan asignado un valor mayor, que deberá aumentar conforme vayan aproximándose al final.

Para poder realizar lo mencionado, deben efectuarse varias pasadas, comparándose los elementos. De esta manera, luego de una primera pasada el número de mayor valor quedará colocado en último lugar. En la segunda vuelta esta posición ya será ignorada, con lo cual se compararán los elementos desde el primero hasta el penúltimo, correspondiéndole ocupar este lugar al mayor de ellos. El proceso continúa hasta que todos los datos queden correctamente ordenados.

Se facilita un programa BASIC pensado para su uso en una máquina Commodore, basado en el diagrama sobre una lista de números introducidos al azar y que deben mostrarse ordenados de mayor a menor.

```
 10 REM CLASIFICACION LISTA NUMEROS
 20 INPUT "ENTRA NUM. ELEMENTOS TABLA:";N
 30 DIM T(N)
 40 FOR X=1 TO N
 50 INPUT "NUMERO"; T(X)
 60 NEXT X
 70 M=N
 80 J=1
 90 FOR P=2 TO M
100 IF T(J)<T(P) THEN GOTO 120
110 J=P
120 NEXT P
130 C=T(M) : T(M)=T(J) : T(J)=C
140 M=M-1
150 IF M>1 THEN GOTO 80
160 FOR R=1 TO N
170 PRINT T(R)
180 NEXT
190 END
```

Portátil notable

La última oferta
El ordenador "de regazo" PX-8 lo fabrica Epson, empresa famosa por sus impresoras matriciales, el portátil HX-20 y el ordenador de gestión para sobremesa QX10. El PX-8 viene con 64 K de RAM, una pantalla de visualización en cristal líquido (LCD), CP/M y varios paquetes de software

Examinamos hoy el PX-8, una máquina "de regazo" con 64 Kbytes de RAM, CP/M y un conjunto de software

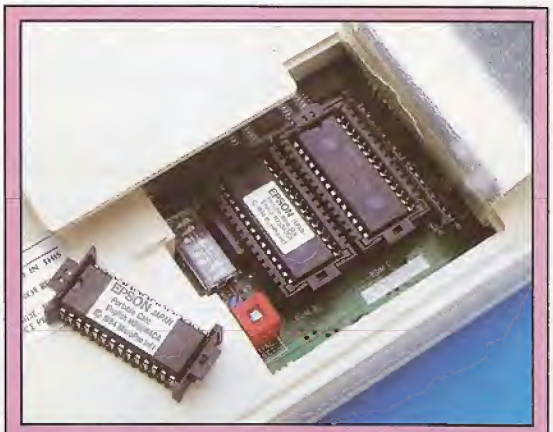
El PX-8 viene en una carcasa del tamaño de una guía telefónica y pesa aproximadamente 2,3 kg. El acabado de la carcasa es en dos tonos de beige, con un asa metálica escamoteable; a primera vista el paquete se parece muy poco a un ordenador. Sin embargo, parte de la carcasa se desliza para dejar al descubierto un teclado completo de ordenador y una pantalla de visualización hasta entonces oculta debajo. La pantalla se libera desplazando un mando etiquetado como "UNLOCK", que también deja al descubierto una grabadora de cintas de microcassette. El panel de la pantalla se puede colocar en cualquiera de 11 posiciones, aunque sólo cinco o seis proporcionan un buen ángulo visual.

El teclado tiene 72 teclas tipo máquina de escribir, codificadas por color para indicar su uso. Las teclas alfanuméricas oscuras están dispuestas de acuerdo al trazado QWERTY estándar en las versiones para Estados Unidos y Gran Bretaña (existe, asimismo, un modelo AZERTY francés a la venta en el resto de Europa), con el signo "£" en el teclado británico reemplazando el "#" (carácter numérico) de la versión norteamericana. (En realidad, desde cualquiera de los teclados se puede acceder a todos los caracteres "internacionales" cambiando los interruptores del PX-8. Este proceso se explica claramente en el manual del usuario.) También hay cuatro teclas de color naranja para control del cursor, teclas Insert, Delete y Home, tres teclas de funciones del sistema (Escape, Pause y Help) y cinco teclas de función programable.

La fabricación del teclado responde a un estándar elevado y es especialmente fácil de utilizar sosteniendo la máquina en el regazo. Sin embargo, el teclado es menos útil cuando el PX-8 está colocado sobre el escritorio, dado que las teclas exigen una presión directa de arriba abajo. Se proporcionan dos patas retráctiles; éstas inclinan la unidad pero no llegan a solventar el problema. Las teclas Caps Lock, Number e Insert son interruptores de posición que se utilizan para cambiar las modalidades de visualización del PX-8: tres pequeños LED rojos indican cuál es la modalidad que está en operación en cada momento.

La documentación Epson es exhaustiva y está muy bien escrita. Se suministran dos gruesos manuales. El primero de ellos es un manual para el usuario de varios cientos de páginas, que abarca la instalación de la máquina, la utilización del hardware y el software y las operaciones de CP/M. Este manual también incluye mapas de memoria, listas completas de los caracteres disponibles y sus códigos, y un programa más bien largo en lenguaje má-

Una gran pantalla
El PX-8 tiene una pantalla LCD de 8 líneas de 80 caracteres que funciona en base a la batería de la máquina. La pantalla proporciona una resolución para visualizaciones gráficas de 480 por 64 pixels



Intercambio de ROM
Deslizando un pequeño panel situado en la cara inferior del PX-8 quedan al descubierto las ranuras que albergan el software basado en ROM. El Portable Wordstar está instalado en la máquina, junto con el sistema operativo CP/M

quina para guardar y cargar la pantalla de gráficos desde disco. El segundo volumen es una excelente guía de referencia de programación en BASIC, que también tiene varios centenares de páginas. Este libro empieza explicando cómo instalar y utilizar el BASIC (suministrado, como el software incluido, en una "cápsula" de ROM), antes de pasar a un claro análisis de la naturaleza de la programación, un examen de las diversas modalidades de visualización del PX-8 y un detallado análisis de todas las instrucciones de BASIC disponibles.

El PX-8 utiliza una CPU CMOS compatible con el Z80. Los chips CMOS (*Complementary Metal Oxide Semiconductor*: semiconductor de óxido metálico complementario) requieren considerablemente menos potencia que los chips de CPU estándares, y este hecho, junto con el empleo del PX-8 de una pantalla LCD de poca potencia, permite que la unidad trabaje enteramente a pilas. Se suministran dos unidades de pilas: una para el consumo principal de potencia y otra como apoyo. Antes de que se pueda utilizar el ordenador se debe cargar la pila, de modo que hay una espera de ocho horas desde que se instala por primera vez la máquina hasta que se halla en condiciones reales de funcionamiento. La unidad principal es recargable y proporciona hasta 15 horas de operación continua antes de que sea necesario volver a cargarla. Según Epson, las expectativas de vida de esta unidad alcanzan a tres o cuatro años.

Una vez que el PX-8 está listo para funcionar, se debe inicializar el sistema operativo. Los pasos necesarios para hacerlo se explican con todo detalle en el manual; éstos implican entrar el día, la fecha y la hora y efectuar algunas tareas "domésticas". Una de éstas es formatear el disco "de RAM". El PX-8 tiene la capacidad de apartar una porción de RAM (que selecciona el usuario entre 9, que es el valor por defecto, y 24 Kbytes) para su empleo como un dispositivo de almacenamiento "en disco". El sistema operativo trata esta zona de la memoria exactamente de la misma forma como lo haría con una unidad de disco externa. Antes de usarlo, el disco de RAM se debe formatear y se debe especificar la cantidad de RAM que utiliza. Asimismo, Epson proporciona una unidad accesoria de disco de RAM, que contiene 120 Kbytes adicionales.

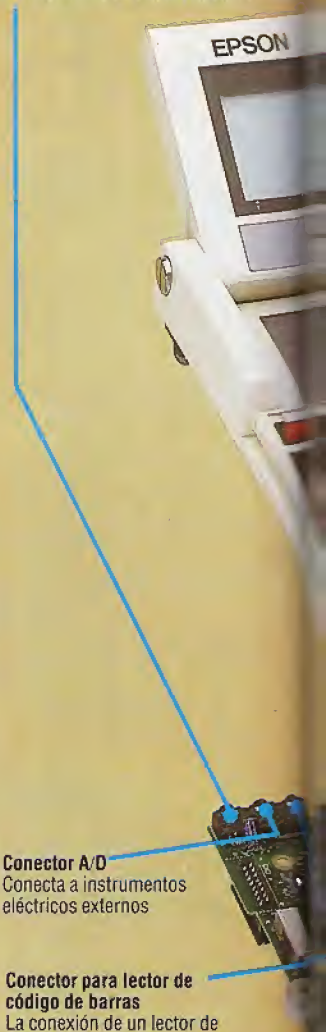
Después de atendidos estos detalles, el PX-8 carga desde la ROM el sistema operativo CP/M y visualiza en la pantalla LCD un directorio de software contenido en ROM y utilidades CP/M en forma de menú. Se puede utilizar software tanto en cassette como en disco o ROM. El software en ROM está retenido en chips EPROM que se colocan en un conector situado debajo de la máquina. El software que se suministra con el PX-8 (Portable Wordstar, Portable Calc y Portable Scheduler) viene en formato de "cápsula", así como el intérprete de BASIC. Para seleccionar una aplicación determinada, se utilizan las teclas del cursor para indicar la opción deseada y luego se pulsa la tecla Return. El programa escogido se carga desde ROM (direccionada por el PX-8 como unidad A y unidad B) en RAM (direccionada como unidad A).

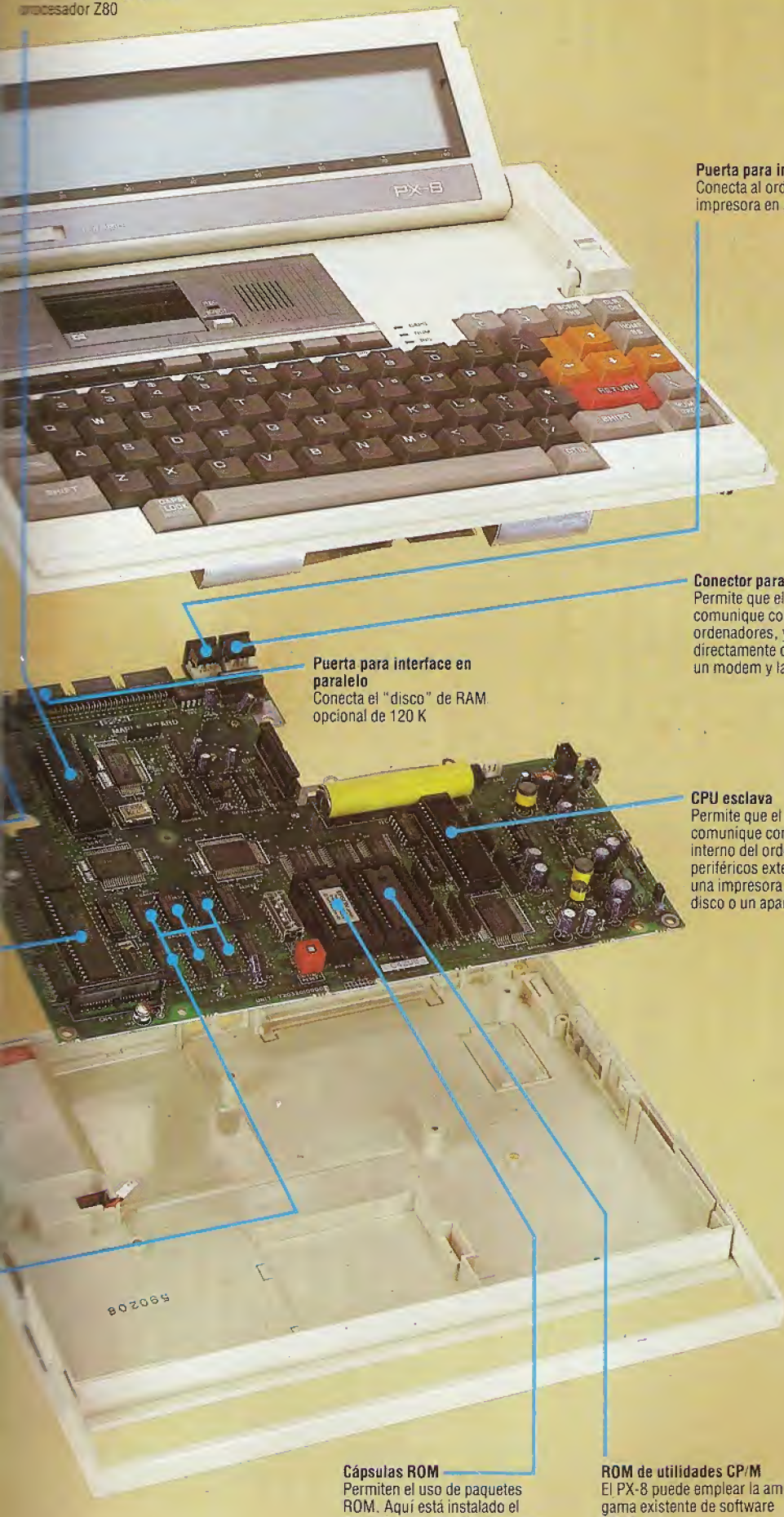
La pantalla LCD proporciona una visualización de 8 líneas de 80 caracteres, con una resolución para gráficos de 480 por 64 pixels. El principal inconveniente de este tipo de pantalla (y, en realidad, el único inconveniente serio de esta excelente máquina) es la lentitud de la visualización. Los caracteres aparecen bastante rápido cuando se los va digitando, pero cualquier borrado (en especial los que implican palabras o frases enteras) es lento.

El software que se suministra con el PX-8 es bastante amplio. Además del procesador de textos, la hoja electrónica y la base de datos ya mencionados, Epson proporciona un programa de telecomunicaciones para utilizar con un modem, y un programa que permite transferir archivos desde el PX-8 a máquinas más grandes, como el QX10 de Epson. Y, puesto que el PX-8 es una máquina CP/M, también puede utilizar el software CP/M ya existente.

El BASIC PX-8 es el Microsoft mejorado por Epson; incluye numeración y renumeración AUTOmática de líneas, un editor completo de pantalla, instrucciones para gráficos y sonido, sentencias que soportan comunicaciones a través de la interface RS232 incorporada, e instrucciones que permiten utilizar la grabadora de microcassette como si se tratara de una unidad de disco (para almacenamiento en acceso directo).

En resumidas cuentas, el Epson PX-8 es un ordenador excelente, ideal, sobre todo, para ejecutivos y periodistas.





Chris Stevens

EPSON PX-8

DIMENSIONES
297×216×48 mm

CPU
CPU CMOS compatible con Z80, 2,4 MHz

MEMORIA
64 K de RAM, 32 K de ROM más 6 K de RAM de video

PANTALLA
Texto: 8 líneas de 80 caracteres
Gráficos: 480×64 pixels

INTERFACES
RS232, en serie, lector de código de barras, entrada analógica

LENGUAJES DISPONIBLES
BASIC Microsoft perfeccionado operando bajo CP/M

TECLADO
Setenta y dos teclas tipo máquina de escribir, formato QWERTY, incluyendo teclas para control del cursor y cinco teclas de función programables. Hay 12 teclas que se pueden utilizar como un teclado numérico

DOCUMENTACION
Dos grandes volúmenes encuadernados con anillas, un manual de operatoria y guía de referencia del BASIC. Los dos son exhaustivos y están bien escritos

VENTAJAS
Pantalla LCD ancha (80 caracteres) que permite seguir fácilmente lo que está sucediendo en la pantalla; el software basado en ROM simplifica la tarea de cargar los programas en la memoria; fácilmente ampliable

DESVENTAJAS
La pantalla LCD muestra sólo 8 líneas y es de manipulación lenta. Aun con una excelente documentación, el CP/M no es un sistema operativo excesivamente sencillo, en especial para el principiante

Control dual

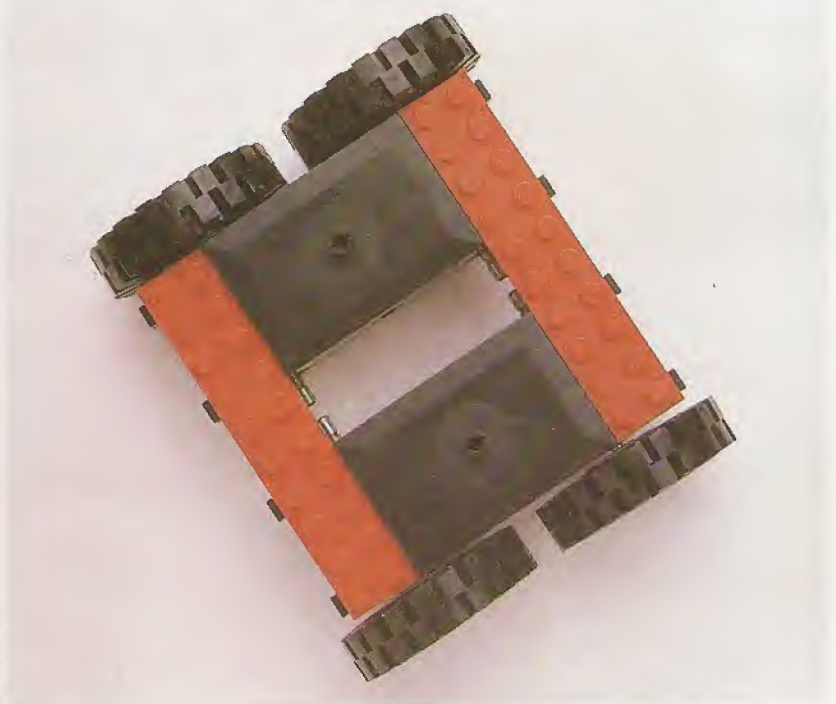
En este capítulo escribiremos el software para controlar un coche Lego conmutando la corriente desde la caja de salida

Si utilizamos dos motores de igual potencia para activar un vehículo, podemos conseguir un control por ordenador en todas las direcciones de movimiento combinando los movimientos hacia adelante y hacia atrás de cada motor. Esto nos permite hacer girar el vehículo además de moverlo hacia adelante o hacia atrás. Existen, en realidad, dos procedimientos para hacer girar un vehículo de motores gemelos; el primero de ellos es sencillamente detener un motor mientras se hace girar el otro. Esto hará que el vehículo gire en un arco, pivotando alrededor de la(s) rueda(s) estacionaria(s). El segundo método consiste en hacer girar un motor hacia atrás mientras el otro gira hacia adelante, mejorando la maniobrabilidad del vehículo puesto que, al girar, éste pivotará alrededor de su eje central.

Podemos controlar cada motor bidireccionalmente mediante la utilización de las cuatro salidas rojas de la caja de salida de poco voltaje (construida en la p. 1054) y conectando el motor derecho a los terminales 0 y 1, y el motor izquierdo a los terminales 2 y 3 (positivos y negativos, respectivamente).

Cada motor se conecta a través de un par de terminales positivos de salida adyacentes, de modo que podemos obtener un control direccional independiente de cada motor. Colocando ahora el número apropiado en el registro de datos, podemos hacer que el vehículo se desplace hacia atrás o hacia adelante, o que gire hacia la izquierda o la derecha.

Uno, mejor que dos
Habiendo descubierto cómo accionar el vehículo Lego hacia adelante y hacia atrás, ahora podemos unir dos de ellos para hacer un *buggy*. Los motores se pueden conmutar individualmente, con lo que el nuevo vehículo es muchísimo más maniobrable que el anterior



Ian McKinnell

El motor del lado derecho (D) irá hacia adelante si la línea 0 está alta y la línea 1 está baja, e irá hacia atrás si la línea 1 está alta y la línea 0 baja. Del mismo modo, el motor del lado izquierdo (I) irá hacia adelante si la línea 2 se pone alta y la línea 3 baja, y hacia atrás en caso contrario. Mediante la combinación de estos movimientos podemos controlar el de todo el vehículo:

Movimiento del vehículo	Motor I	Motor D	Patrón de bits	N.º en el REGDAT
APAGADO	APAGADO	APAGADO	0000	0
ADELANTE	ADELANTE	ADELANTE	0101	5
ATRAS	ATRAS	ATRAS	1010	10
PIVOTE IZQ.	ADELANTE	ATRAS	0110	6
PIVOTE DER.	ATRAS	ADELANTE	1001	9
ARCO IZQ.	ADELANTE	APAGADO	0100	4
ARCO DER.	APAGADO	ADELANTE	0001	1

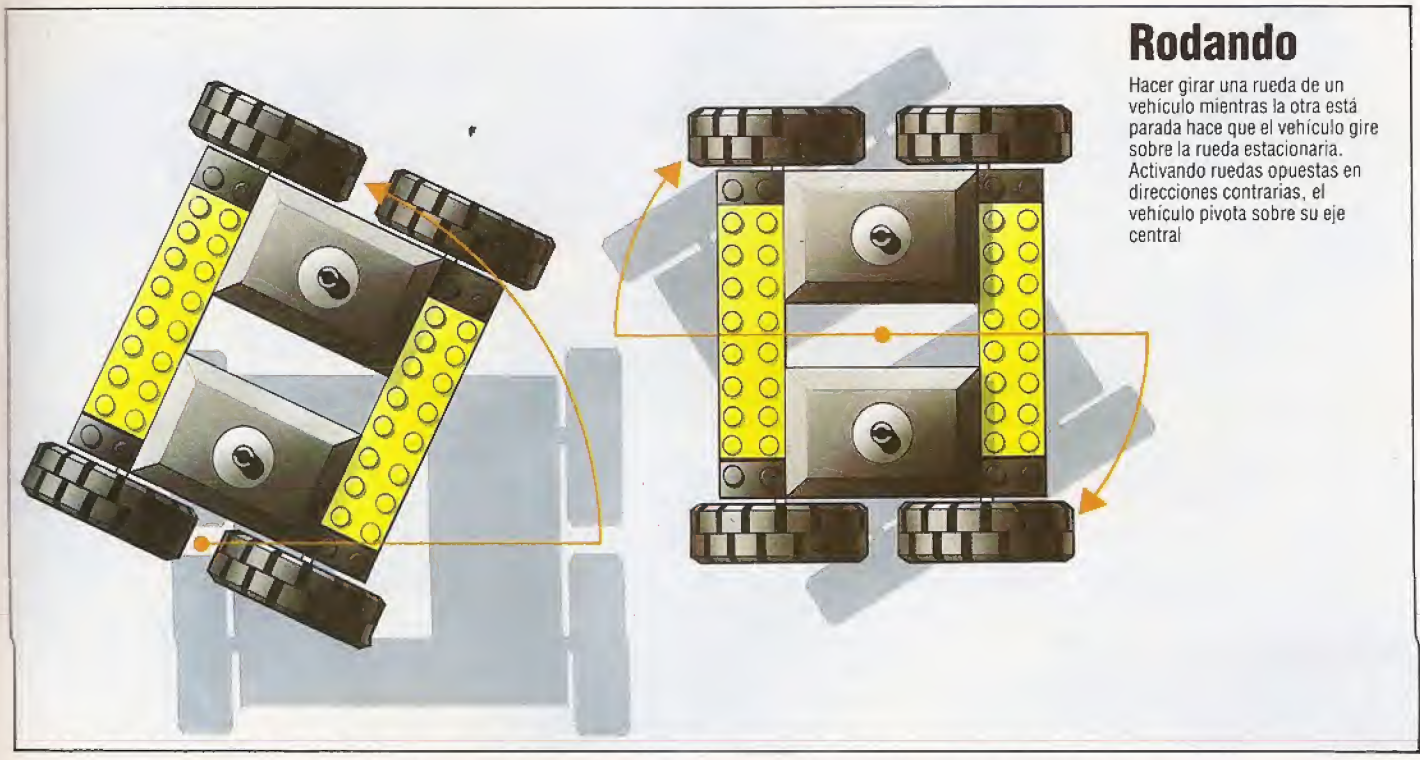
El siguiente programa nos permite controlar el vehículo desde el teclado utilizando "T" para marcha adelante, "B" para marcha atrás, "F" para girar a la izquierda, y "H" para girar a la derecha. Si no se pulsa ninguna tecla, el vehículo se detiene.

BBC MICRO

```
 10 REM MOTORES GEMELOS BBC
 20 RDD=&FE62:REGDAT=&FE60
 30 ?RDD=255
 40 REPEAT
 50 A$=INKEY$(10)
 60 PROCleer—teclado
 70 UNTIL A$="X"
 80 ?REGDAT=0
 90 END
1000 DEF PROCleer—teclado
1010 IF A$="" THEN ?REGDAT=0
1020 IF INKEY(-36)=-1 THEN ?REGDAT=5
1030 IF INKEY(-101)=-1 THEN ?REGDAT=10
1040 IF INKEY(-68)=-1 THEN ?REGDAT=6
1050 IF INKEY(-85)=-1 THEN ?REGDAT=9
1060 ENDPROC
```

COMMODORE 64

```
 10 REM MOTORES GEMELOS CBM 64
 20 RDD=56579:REGDAT=56577
 25 POKE650,128:REM REPETIR MODO TECLA
 30 POKE RDD,255
 40 GETA$
 50 GOSUB1000:GOTO70
 60 POKEREGDAT,0
 70 IF A$ <> "X" THEN FOR I=1TO100:NEXT:GOTO40
 80 POKE REGDAT,0
 90 END
1000 REM COMPROBAR ENTRADA S/R
1005 IF A$="" THEN POKE REGDAT,0
1010 IF A$="T" THEN POKE REGDAT,5
1020 IF A$="B" THEN POKE REGDAT,10
1030 IF A$="F" THEN POKE REGDAT,6
1040 IF A$="H" THEN POKE REGDAT,9
1050 RETURN
```



Rodando

Hacer girar una rueda de un vehículo mientras la otra está parada hace que el vehículo gire sobre la rueda estacionaria. Activando ruedas opuestas en direcciones contrarias, el vehículo pivota sobre su eje central

Kevin Jones

En ambas versiones, el vehículo sólo se moverá mientras se mantenga pulsada una tecla. Apenas se suelte ésta, los motores se pararán al colocarse un cero en el registro de datos. En los dos casos la salida del programa se realiza pulsando "X".

En la versión del programa para el BBC, el procedimiento LEER-TECLADO nos permite verificar el teclado directamente, en vez de leer el buffer del teclado, mediante la utilización de INKEY. Ello nos proporciona una mejor respuesta en el control del vehículo. La versión para el Commodore 64 en primer lugar activa la repetición automática del teclado, de modo que, si se mantiene pulsada una tecla, se seguirán enviando caracteres al buffer del teclado para ser leídos por la instrucción GET. Lamentablemente, no existe ninguna forma de leer el teclado directamente y, por consiguiente, un control exacto es más difícil que en el BBC Micro. La sensibilidad se puede mejorar limpiando el buffer del teclado justo antes de leerlo. Insertando la siguiente línea en la versión del programa para el Commodore 64 se conseguirá esto.

```
35 GET J$:IF J$ <> "" THEN35
```

Además, el GOTO al final de la línea 70 se debe cambiar por GOTO35.

La velocidad a la cual se repite una tecla cuando se la mantiene pulsada puede plantear un problema en las dos versiones de este programa. Si el bucle del programa principal se ejecuta más rápidamente que el tiempo de repetición de una tecla, entonces cuando la rutina tenga que volver a comprobar si se ha pulsado alguna, pensará que no se está pulsando ninguna. Ello provocará un rápido encendido y apagado del motor puesto que la salida alterna rápidamente entre los valores para la dirección elegida y cero. En ambas versiones del programa se ha solventado este problema añadiendo código para aumentar el tiempo de ejecución del bucle del programa principal. En la versión para el BBC, la utilización de INKEY$(10) hace que el ordenador quede "colgado" durante 10 centésimas de segundo, a la espera de una entrada, antes de continuar. En la versión para el Commodore 64 se ha agregado un pequeño bucle de retardo en la línea 70. Los valores de este retardo se hallaron mediante un proceso de ensayo y error, y dependen del lapso necesario para ejecutar una pasada de la rutina. Cuando escriba sus propios programas, quizá se encuentre con que el tiempo de ejecución de la rutina excede a la velocidad de repetición de tecla; de no ser así, inserte en su código un pequeño retardo.

Ahora que hemos obtenido control sobre los movimientos de nuestro vehículo, es interesante diseñar un programa que "memorice" una secuencia de movimientos y los reproduzca. Para hacerlo, podemos valernos de una matriz de dos dimensiones que grabe la dirección y el tiempo consumido en cada una de las maniobras efectuadas. La primera parte de un programa de esta clase será la misma que las que ya hemos dado, pero la segunda reproducirá los datos almacenados. Los datos se almacenarán en una matriz DR(), donde DR(C,1) almacena la dirección y DR(C,2) el tiempo que consume cada movimiento. Cada vez que se selecciona una nueva dirección se utiliza un nuevo elemento de la matriz. Esta condición se indica mediante un cambio en el contenido del registro de datos. Se emplea un contador, C, para indexar la matriz.

BBC MICRO

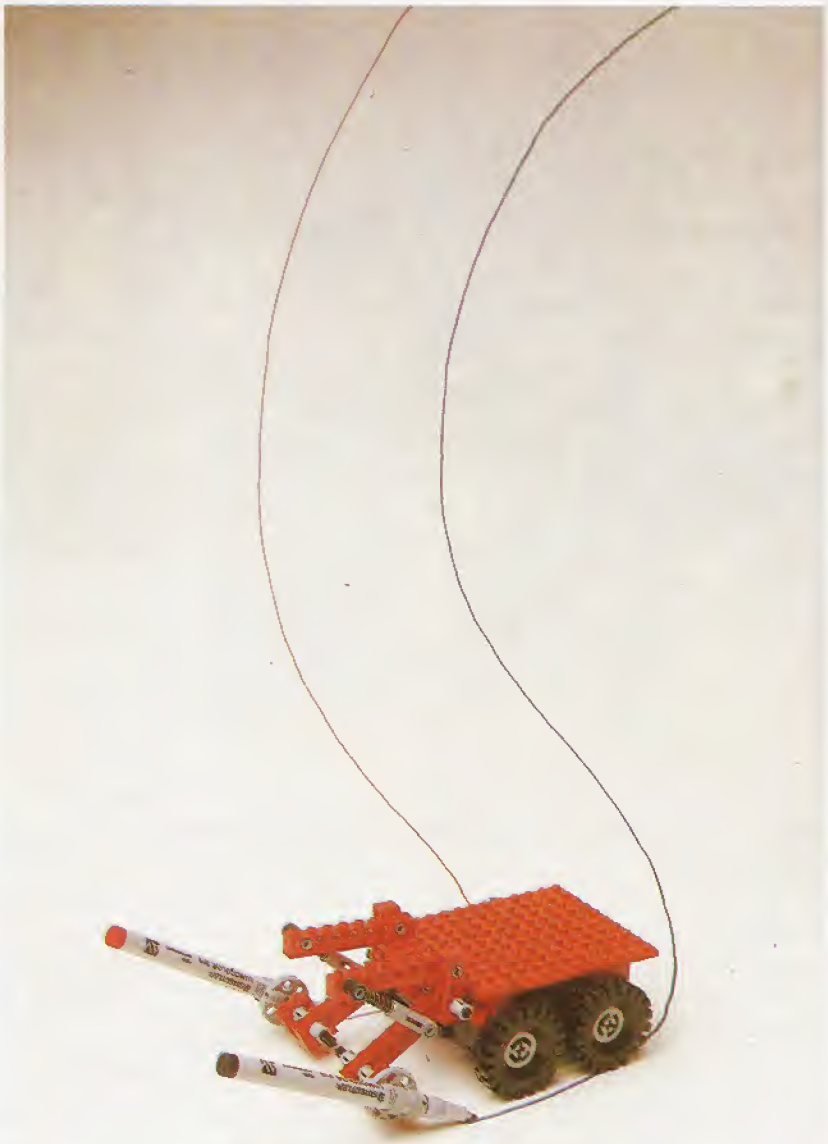
```
1000 REM MEMORIA DE MOVIMIENTOS BBC
1010 RDD=&FE62:REGDAT=&FE60
1020 DIM DR(100,2)
1030 ?RDD=255:C=1:REM INICIALIZAR CONTADOR
1040 REPEAT
1050 A$=INKEY$(10)
1060 PROCleer—teclado
1070 UNTILA$="X"
1080 ?REGDAT=0
1090 DR(C-1,2)=TIEMPO
1100 REPEAT A$=GET$
1110 UNTIL A$="C"
```

```
1120 REM REPRODUCIR DATOS
1130 FOR I=1TOC
1140 ?REGDAT=DR(I,1)
1150 TIEMPO=0
1160 REPEAT UNTIL TIEMPO > =DR(I,2)
1170 NEXT I
1180 END
1190 :
1200 DEF PROCleer—teclado
1210 IF A$="" THEN ?REGDAT=0
1220 IF INKEY(-36)=-1 THEN ?REGDAT=5
1230 IF INKEY(-101)=-1 THEN ?REGDAT=10
1240 IF INKEY(-68)=-1 THEN ?REGDAT=6
1250 IF INKEY(-85)=-1 THEN ?REGDAT=9
1260 PT=?REGDAT
1270 IF PT <> DR(C-1,1)THEN PROCsumar—datos
1280 ENDPROC
1290 :
1300 DEF PROCsumar—datos
1310 DR(C-1,2)=TIEMPO:REM ALMACENAR ULTIMO
     TIEMPO
1320 TIEMPO=0:REM COMENZAR NUEVO TIEMPO
1330 DR(C,1)=PT:REM ALMACENAR ESTADO PUERTA
1340 C=C+1:REM INCREMENTAR CONTADOR
1350 ENDPROC
```

COMMODORE 64

```
  10 REM MEMORIA DE MOVIMIENTOS CBM 64
  15 DIMDR(100,2): REM MATRIZ DE DIRECCION
  20 RDD=56579:REGDAT=56577
  25 POKE650,128:REM ESTABLECER MODO REPETICION
     TECLA
  30 POKERDD,255:REM TODAS SALIDA
  35 C=1:REM INICIALIZAR CONTADOR
  40 GETA$
  50 GOSUB1000:REM COMPROBAR ENTRADA
  70 IF A$ <> "X" THEN FOR I=1TO200:NEXT:GOTO40
  80 POKE REGDAT,0:REM APAGADO
  85 DR(C-1,2)=TI-T:REM ENTRAR ULTIMO TIEMPO
  90 STOP:REM PULSAR "CONT" PARA CONTINUAR
  95 REM REPRODUCIR DATOS
 100 FORI=1TOC
 110 POKEREGDAT,DR(I,1)
 120 T=TI
 130 IF(TI-T) < DR(I,2)THEN130
 140 NEXT
 150 END
 999 :
1000 REM COMPROBAR ENTRADA S/R
1005 IFA$=""THEN POKEREGDAT,0
1010 IFA$="T"THEN POKEREGDAT,5
1020 IFA$="B"THEN POKEREGDAT,10
1030 IFA$="F"THEN POKEREGDAT,6
1040 IFA$="H"THEN POKEREGDAT,9
1045 PT=PEEK(REGDAT)
1050 IF PT <> DR(C-1,1)THENGOSUB1500
1498 RETURN
1499 :
1500 REM AÑADIR DATOS A MATRIZ
1510 DR(C-1,2)=TI-T:REM AÑADIR ULTIMO TIEMPO
1520 T=TI:REM TOMAR NUEVO TIEMPO
1530 DR(C,1)=PT:REM ENTRAR CONTENIDO ACTUAL
     PUERTA
1540 C=C+1:REM INCREMENTAR CONTADOR
1999 RETURN
```

Este programa le permite al usuario desplazar el vehículo controlándolo desde el teclado. Dado que cada movimiento se graba como una dirección y un intervalo de tiempo, cualquier error introducido en el cronometraje de cada movimiento producirá errores en la reproducción. Estamos penetrando en el área de la informática en tiempo real, en donde la estructura del programa y el tiempo de ejecución se pueden convertir en importantes factores.



Ejercicios

Ahora que podemos controlar el movimiento de un vehículo en todas las direcciones, surgen muchas posibilidades para realizar breves ejercicios de programación. Probablemente se le ocurrirán muchas ideas; aquí le proporcionamos algunas:

1) Intente calibrar su vehículo. ¿Durante cuánto tiempo ha de estar el número correspondiente en el registro de datos para hacer que el vehículo se desplace un metro hacia adelante o hacia atrás, o efectúe un giro de 90°?

2) Diseñe un recorrido con obstáculos para su vehículo y, utilizando los programas ofrecidos como base, escriba uno que le permita "enseñarle" a realizar el recorrido bajo el control del teclado. Una vez que lo haya guiado a través de su desplazamiento, el programa deberá invertirse, guiando el vehículo de vuelta hasta su punto de partida.

3) Conecte a la caja buffer cuatro interruptores que le permitan controlar el vehículo externamente desde la puerta para el usuario.

Movimientos con memoria

Es bastante sencillo escribir un programa para controlar un *buggy* que acepte direcciones desde el teclado y mueva el coche en consecuencia. No es mucho más difícil ampliar el programa de modo que almacene las instrucciones del operador y se las reproduzca después al vehículo, produciendo, por consiguiente (en teoría), el patrón de movimiento anterior. Comparando el original con la supuesta réplica se obtiene una medida de los problemas de software que surgen al tratar con el mundo real: el ordenador trabaja con números y tiempos exactos en un modelo simplificado de un universo perfecto, sin dejar lugar para la inercia, las pérdidas por fricción, las superficies irregulares ni la ingeniería de baja tolerancia. A la luz de esta experiencia, el rendimiento de las tortugas guiadas por LOGO es notable

Fuerzas invasoras

Esta vez nos referiremos a la versión original de "Space invaders", popular y pionero juego recreativo de Atarisoft

Casi todos los ordenadores personales que se comercializan actualmente disponen de una versión del *Space invaders* (Invasores del espacio). El juego ha llegado a hacerse tan conocido que a menudo se utiliza como término genérico, hasta el punto de que se suele decir que alguien que esté jugando a cualquier juego recreativo está "jugando a los invasores del espacio".

En 1978, cuando se lanzó este juego, produjo rápidamente una fiebre de proporciones casi epidémicas. Los padres comenzaron a preocuparse por el hecho de que los niños invirtieran todo su tiempo y su dinero deambulando por atestadas salas recreativas. Lo que los celosos guardianes no comprendían era que estos niños en realidad estaban investigando el futuro.

Se puede decir que *Space invaders* cambió la forma en que la sociedad veía a los ordenadores. Antes de que apareciera el juego explotando las capacidades gráficas del microprocesador, los ordenadores se consideraban poco dignos de confianza, siendo el ejemplo clásico el paranoico HAL de la película *2001: una odisea del espacio*.

Space invaders fue el precursor de todos los juegos por ordenador del género de "marcianitos". Desde entonces se han producido literalmente centenares de juegos en los que un héroe o una heroína han tenido que enfrentarse a hordas de deleznables seres atacantes sólo con la velocidad de un disparador (y tres vidas) como ayuda.

Es una verdad indiscutible que actualmente *Space invaders* está evidenciando su edad. Para los estándares de hoy en día, el juego es muy simple; pero, a pesar de ello, de todo el software que se ha producido ninguno ha atrapado en tal medida la imaginación del público. El jugador controla una base de láser móvil, que se utiliza para disparar contra formaciones masivas de extraterrestres invasores que van descendiendo amenazadoramente por la pantalla hacia la superficie de la Tierra. Se pierde una "vida" si la base de láser es alcanzada por el fuego de los extraterrestres o si éstos llegan hasta la parte inferior de la pantalla.

Existen varias diferencias entre el juego original y las versiones existentes para ordenadores personales. En vez de aparecer de la nada, ahora las hordas invasoras emergen de un gran cohete situado a la izquierda de la pantalla. Los invasores propiamente dichos son de colores más brillantes y los sprites que los representan son más complejos. Las barreras defensivas, tras las cuales se podía ocultar la base de láser en la versión recreativa, ahora ya no están, y los invasores han de recorrer una distancia más corta para alcanzar la parte inferior de la pantalla. Pero hay un factor que se ha mantenido constante: el amenazador sonido de "latidos de corazón" que acompaña el descenso de los extraterrestres. Éste se vuelve más insistente a medida que los invasores se van acercando y sirve para generar una alta dosis de adrenalina, que probablemente es la principal razón del colosal éxito del juego. Otra característica que comparten las versiones recreativas y para ordenadores personales es el premio "misterio" que se concede cuando el jugador consigue darle a alguno de los platillos volantes que ocasionalmente atraviesan la pantalla de izquierda a derecha.

Space invaders ha conseguido mantener su atractivo durante seis años. A pesar de la existencia de software considerablemente más sofisticado, continúa siendo un juego emocionante y sumamente divertido; es realmente un "clásico del software".

Space invaders: Para todos los ordenadores Atari
Editado y distribuido por: Atari, AUDELEC, Compás de la Victoria, 3, 29012 Málaga, España
Autores: Atari
Palanca de mando: Necesaria
Formato: Cartucho

Liz Dixon

Punteros útiles

Ahora introduciremos mejoras en los programas de definición de caracteres, centrándonos en su almacenamiento y recuperación

Ahora que nuestros programas para definir caracteres están listos y en funcionamiento (véanse pp. 1052 y 1068), vale la pena dedicar un poco de tiempo a comparar las tres versiones y, también, a mejorarlas. La versión para el Commodore se escribió primero, porque es la más difícil de programar de las tres. Esta versión se tradujo luego, virtualmente línea por línea, para las otras dos máquinas. En parte debido a esta traducción y a que el espacio era limitado, el formateado de la pantalla es rudimentario, y no se utiliza ni color, ni sonido, ni gráficos en alta resolución. Es evidente que, en consecuencia, se pueden introducir mejoras en todos estos sentidos, pero no nos ocuparemos de éstas aquí.

Dejando aparte las cuestiones relativas a la eficacia de la programación (no realmente esencial en este programa, dado que no hay tareas que dependan de la velocidad), nos concentraremos en la interface para el usuario: instrucciones, ayudas, teclas de instrucciones y facilidades.

En el programa no hay instrucciones, básicamente porque el listado debía caber en una sola página del curso. Al principio de la ejecución se podría imprimir en la pantalla una página de instrucciones, y en la visualización de la pantalla principal probablemente hay sitio para incluir algunos recordatorios abreviados: una visualización del movimiento del cursor, quizá, o resúmenes de una palabra de las teclas de instrucciones. Ello eliminaría en gran medida la necesidad de una página de ayuda.

La elección de las teclas de instrucciones se podría mejorar. En el BBC Micro y el Spectrum el cursor se mueve por la ventana mediante las teclas normales de control del cursor, mientras que en el Commodore se utilizan las teclas de función sin SHIFT. Esto es mucho más adecuado para la programación de la versión del Commodore, dado que los códigos ASCII de las ocho teclas de función son consecutivos entre 133 y 140, pero el trazado de las teclas no es exactamente ergonómico y no se repiten, a diferencia de las teclas del BBC y el Spectrum. Esto último se puede modificar en el Commodore mediante POKE650,128, pero lo que no se puede es hacer que las teclas de función resulten más fáciles de utilizar, de modo que tal vez desee devolverles el control del cursor a las teclas correspondientes a éste.

Otra posible mejora es la elección de la estrategia para el movimiento del cursor. Tal como está escrita, ésta simplemente desautoriza por ilegal cualquier instrucción que haga que el cursor salga de la ventana. La alternativa consiste en hacer aparecer el cursor por la derecha de la pantalla cuando desaparezca por la izquierda, y viceversa, procediendo de modo similar para el movimiento vertical. Esto es fácil de programar, pero exige más código que el de las comprobaciones simples utilizadas en la subrutina 3500.

Las instrucciones proporcionadas son las mínimas necesarias, y se podrían ampliar. En el Spectrum y en el BBC, guardar y cargar (SAVE y LOAD) juegos de caracteres se podría incluir como instrucciones, y en las tres versiones sería muy útil poder copiar la definición de un carácter en la de otro, de modo que, por ejemplo, CHR$(N) y CHR$(N+1) representaran el mismo carácter. Tal vez se desee una salida impresa del nuevo juego de caracteres, por lo que también se podría agregar una opción para impresora. La sencilla estructura modular del programa hace que estas instrucciones se puedan añadir de forma bastante directa.

El SAVE del Commodore 64

Un problema exclusivo del BASIC Commodore es que la instrucción SAVE parece referirse sólo a la zona entera para programas en BASIC, mientras que los BASIC de las otras dos máquinas permiten que el usuario especifique la zona de memoria que desea guardar. La instrucción LOAD del Commodore, sin embargo, sí permite cargar archivos en cualquier zona deseada, de modo que si pudiéramos solucionar el problema del SAVE podríamos almacenar y recuperar los nuevos juegos de caracteres.

La instrucción SAVE del Commodore está señalada en la zona para programas en BASIC mediante dos punteros de dirección: TXTTAB (en las posiciones 43 y 44) y VARTAB (en las 45 y 46). El primero, TXTTAB, apunta el comienzo de la zona para programas en BASIC (por lo general, a partir de la posición 2048), mientras que VARTAB apunta al comienzo de la zona para variables del BASIC; puesto que ésta comienza donde terminan los programas en BASIC, en realidad VARTAB señala el final de la zona para programas en este lenguaje. Si cambiáramos estos punteros de modo que indicaran el comienzo y el final del nuevo juego de caracteres y generásemos luego una instrucción SAVE, solucionaríamos el problema.

Antes de hacer esto, no obstante, deberíamos reconsiderar la posición del propio juego de caracteres. La subrutina de la línea 61000 (véase p. 1053) copia el juego de caracteres de ROM en un bloque de RAM de dos Kbytes que empieza en 14336, y la

Ian McKinnell

línea 50 coloca el puntero al tope de la memoria al final de este bloque, para evitar que se machaque el BASIC. De esta manera, con el objeto de proteger dos Kbytes estamos recortando dos tercios de la memoria para el usuario. Esto no supone ningún problema para la ejecución del programa generador de caracteres, pero sería una fuente potencial de dificultad si cargáramos el nuevo juego de caracteres en esa dirección para que lo utilice un programa de aplicaciones que necesite más de 12 Kbytes de memoria para el usuario. Lamentablemente, la mecánica del sistema operativo impide que coloquemos el juego de caracteres en un lugar superior de la memoria; si esto fuera posible, podríamos ponerlo arriba, en los dos Kbytes más altos de la memoria para el usuario, o en la zona para programas especiales, desde 49152 en adelante. La solución será colocar el juego de caracteres lo más abajo posible, ¡y desplazar el BASIC por arriba de él! Esto se puede hacer ajustando el contenido de los punteros TXTTAB, pero no desde un programa en BASIC, y debe efectuarse antes de cargar en la memoria el programa generador de caracteres.

La secuencia de acciones es, por consiguiente:
1) Cargar (LOAD) y ejecutar (RUN) el Programa 1. Éste imprime en la pantalla las instrucciones de reubicación necesarias, de modo que puedan ejecutarse en modalidad directa, pulsando Return.
2) Cargar (LOAD) el programa generador de caracteres y efectuar las siguientes modificaciones:

```
61100 CGEN=53248:NCGEN=2048
61500 POKE PO,(PEEK(PO)AND240)OR2
```

y borrar la línea 50.
3) Guardar (SAVE) esta nueva versión.
4) Cargar (LOAD) y ejecutar (RUN) el generador de caracteres exactamente como antes.
5) Cuando termine, cargue y ejecute el Programa 2. Al igual que el Programa 1, imprime en la pantalla instrucciones para que usted las ejecute.
6) El Programa 2 estableció los punteros TXTTAB y VARTAB, de modo que SAVE"nombrearchivo" guarda la zona completa de 2 Kbytes para el juego de caracteres entre 2048 y 4097. En el futuro, para ejecutar el programa generador de caracteres debe repetir esta secuencia, con la excepción del paso 2.

Cuando desee recuperar el juego de caracteres, debe cargar y ejecutar el Programa 1, para desplazar el BASIC hacia arriba de la memoria, y después cargar el juego de caracteres así:

```
LOAD"nombrearchivo",DN,1
```

donde DN (número de dispositivo) es igual a uno para cuando se emplee cassette y es ocho para la unidad de disco. El ",1" al final de la instrucción es conocido como la dirección secundaria, y es la forma de Commodore de enviarles parámetros de instrucciones a los dispositivos periféricos. Por tanto, significa que el archivo se ha de cargar en el lugar de la memoria del cual se guardó, en vez de ser dirigido por el puntero TXTTAB a la zona normal para programas en BASIC. Esto es posible porque cuando se guarda (SAVE) un archivo, el sistema operativo guarda la dirección de comienzo de RAM como el primer dato del archivo. Cuando se emplea la instrucción LOAD sin más, la dirección de comienzo del archivo se ignora, atendiendo en cambio a la dirección que señala TXTTAB.

valor de X=	bits 3,2,1,0	posición a la que se apunta
0	0000	0
2	0010	2048
4	0100	4096
6	0110	6144
8	1000	8192
10	1010	10240
12	1100	12288
14	1110	14336

Tablas de valores
POKE 53272,(PEEK(53272)AND240)OR X obliga al chip de visualización a mirar a la zona de RAM que contiene los caracteres redefinidos. La tabla muestra los valores de X que establecen la dirección de comienzo del bloque de RAM.

Una vez reubicado el BASIC y cargado el nuevo juego de caracteres, debe hacer que el sistema operativo apunte hacia éste; esto se explica en la tabla y se muestra en la nueva versión de la línea 61500.

```
199 REM**********************************
200 REM*              PROGRAMA 1          *
201 REM*       EJECUTE ESTE PROGRAMA      *
202 REM*  LUEGO PULSE RETURN DOS VECES    *
203 REM*       Y EL BASIC PASARA A 4096    *
204 REM**********************************
300 PRINT CHR$(147):PRINT:PRINT
400 PRINT"POKE43,0:POKE44,16:POKE-
    45,3:POKE46,16"
500 PRINT"POKE4096,0:POKE4097,0:POKE
    4098,0:CLR:NEW"
600 PRINT CHR$(19)
```

```
199 REM**********************************
200 REM*              PROGRAMA 2          *
201 REM*       EJECUTE ESTE PROGRAMA      *
202 REM*  LUEGO PULSE RETURN DOS VECES.   *
203 REM* SE RESTAURAN LOS PUNT. DE BASIC  *
204 REM**********************************
300 PRINT CHR$(147):PRINT:PRINT
400 PRINT"POKE43,0:POKE44,8:POKE-
    45,1:POKE46,16"
500 PRINT"POKE4096,0:POKE4097,0:POKE
    4098,0:CLR"
600 PRINT CHR$(19)
```

Con su pareja

Después del estudio del direccionamiento indexado, en esta ocasión vamos a analizar las operaciones aritméticas y las subrutinas comparativas

La lección anterior trató del direccionamiento indexado en el 6809. En este modo de direccionamiento, la dirección efectiva especificada por DESPL,X se forma como la suma del desplazamiento (que puede ser una constante o el contenido de una posición de memoria) y el valor que contenga el registro índice especificado (en este caso, el X). Vimos que en algunos casos comunes, el desplazamiento puede ser cero, en cuyo caso podemos escribir ,X (también funcionaría 0,X). En algunas circunstancias especiales podemos servirnos de acumuladores A,B o D para expresar el desplazamiento (p. ej., B,X). Analizamos también la manera de hacer más fácil el uso del modo autoincremento y autodecremento, para uno de los empleos más frecuentes del direccionamiento: el de recorrer una tabla de valores. Tales modos incrementan el registro en una o en dos unidades una vez ejecutada la instrucción (,X+ y ,X++), o bien decrementan éste de la misma manera pero antes de ejecutar la instrucción (,−Y y ,−−Y).

Vamos ahora a examinar brevemente la forma como podemos emplear el direccionamiento indexado para realizar sencillas operaciones aritméticas con valores contenidos en los registros índice gracias a la instrucción LEA (*Load Effective Address*: cargar la dirección efectiva). Las diversas instrucciones normales de tipo aritmético sólo funcionan en los acumuladores, no en los restantes registros. Sin duda que podemos transportar el contenido del registro índice al acumulador D, realizar la operación y devolver el resultado al registro, pero éste es un procedimiento lento y engorroso. La instrucción LEA (sólo aplicable a los registros X,Y,S y U) hará los cálculos necesarios de la dirección y cargará el valor efectivo de ésta. Como quiera que es frecuente cargar el contenido de una dirección efectiva, nos encontramos ante una alternativa digna de tener en cuenta.

Veamos un ejemplo. La instrucción

```
LEAX    −1,X
```

calculará la dirección efectiva sumando −1 al valor actual del registro X. Esta dirección se carga en X, decrementando efectivamente el valor previo del registro. Pero la instrucción no sólo sirve para esto. Puede también utilizarse, por ejemplo, para calcular una sola vez una dirección y guardar el resultado, con objeto de evitar repetir el mismo cálculo varias veces.

Se puede igualmente realizar cierto número de operaciones aritméticas sobre el registro X mediante ABX (*Add* B *to* X: sumar B a X), una instrucción que sirve para sumar sin signo valores contenidos en B a los contenidos en X. Pero no resulta tan útil como LEA.

Subrutinas

Una subrutina es un fragmento autosuficiente de código máquina que pueda ser llamado por el programa principal (o por otra subrutina) para realizar una tarea específica. Una vez efectuada ésta, se trasfiere automáticamente el control al programa o rutina que la llamó (*call*) para continuar con la instrucción inmediatamente siguiente a la de la llamada. Hay, al menos, tres razones para emplear subrutinas:

1) Porque nos ahorramos el escribir el mismo fragmento cada vez que lo necesitemos. Es mucho mejor escribirlo una sola vez como subrutina y llamarlo cada vez que sea necesario su uso.
2) Porque podemos hacernos con toda una "subrutinoteca" y utilizar las rutinas en programas diferentes.
3) Porque así se subdivide el programa en trozos más pequeños y más manejables.

Lo que hay que tener más en cuenta en las subrutinas del assembly es que emplean los mismos registros que el programa que las llama. Uno de los errores más comunes en la programación de código máquina consiste en que, una vez almacenado un valor en alguno de los registros, el programa puede llamar a una subrutina que altera dicho valor y el programador no lo tiene en cuenta en el momento de devolver el control. Por ello es esencial conocer y documentar los registros que utiliza la subrutina. Es especialmente importante salvar los contenidos de los registros que van a ser utilizados por la subrutina y restaurarlos una vez que la subrutina haya completado su tarea.

Más adelante nos detendremos a examinar cómo se emplean las pilas tanto para salvar tales datos como para pasar parámetros (valores y direcciones) a la subrutina. Por el momento asumiremos que la subrutina emplea los mismos datos que el programa que la llama (variables globales) y que cualquier otro valor que necesite se encontrará en los registros. La llamada de una subrutina se hace mediante una de estas instrucciones:

- BSR (*Branch to SubRoutine*: bifurcación a subrutina)
- JSR (*Jump to SubRoutine*: salto a subrutina)

La orden BSR provoca una bifurcación relativa, es decir, encuentra la subrutina en un cierto desplazamiento respecto al valor actual del contador de programa. Se suele utilizar esta instrucción para subrutinas escritas formando parte del programa.

La instrucción JSR llama a la subrutina especificando una dirección determinada. Se emplea cuando la subrutina se retiene en la ROM o bien cuando

se dispone de una biblioteca de rutinas situada siempre en el mismo lugar de la memoria, por ejemplo formando parte del sistema operativo de disco.

Cuando el procesador encuentra una instrucción BSR o JSR, el valor en curso del contador de programa es colocado (*push*) en la pila del sistema utilizando el registro S (puntero o índice de la pila). Si esta subrutina que llamamos emplea el registro S para cualquier otro objeto aparte del de llamar a su vez a otra subrutina, nunca deberemos olvidarnos de restituir el valor correcto. El resultado del cálculo de la dirección de la subrutina (para el caso de BSR) se carga en el contador de programa. De esta manera la siguiente instrucción que hay que ejecutar coincidirá con la primera de la subrutina. Asegúrese, por lo tanto, de que la subrutina comienza con una instrucción y no con algún byte de datos.

La subrutina debe concluir con una instrucción RTS (*ReTurn from Subroutine*: volver de la subrutina), que tiene por efecto el de extraer (*pull*) de la pila el valor anterior para reponerlo en el contador del programa. La ejecución de éste se reanudará allí donde quedó interrumpida por causa de la llamada a la subrutina.

El programa de ejemplo que damos aquí resulta algo más complicado de lo acostumbrado, pero se puede hacer más manejable si empleamos una subrutina. Se trata de buscar una tabla que contiene series o cadenas (*strings*) de desigual longitud y extraer un valor que se corresponda con una determinada serie. Se han colocado las series de la manera habitual: se inician con un byte que indica la longitud de la serie y seguidamente van los caracteres que la componen para finalizar con una dirección de 16 bytes que corresponde a esa cadena.

El final de la tabla se señala con una cadena de longitud cero, o sea, se encuentra el cero donde debería encontrarse el byte de longitud. Supondremos que la dirección de comienzo de la tabla se sitúa en $10, y que la dirección de la cadena cuya correspondencia buscamos está en $12. Si se encuentra la pareja en la tabla, la dirección correspondiente se colocará en $14. Pero si no es hallada, se pondrá a cero tanto $12 como $14.

Emparejamiento de series

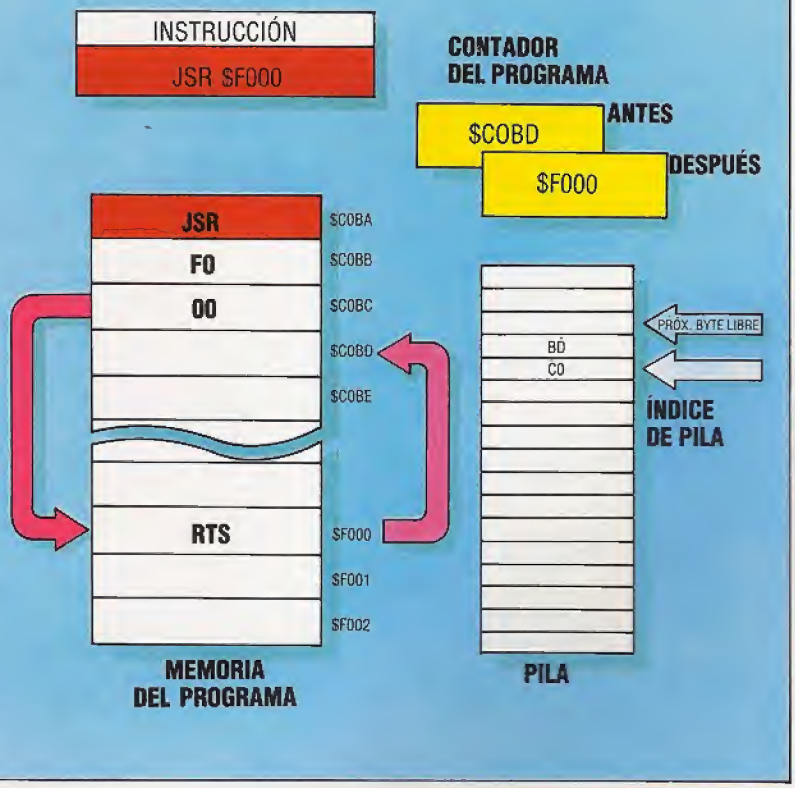
El emparejamiento de series o cadenas es una tarea que se da en múltiples ocasiones, especialmente cuando se manejan accesos o variables alfanuméricas por medio del intérprete BASIC: cada identificador (o nombre de la variable) debe ser sustituido por las direcciones en las que está almacenado el valor de esa variable.

Es fácil dividir el problema en dos partes: hay que recorrer la tabla hasta que encontremos o bien la cadena que buscamos o bien el final de la tabla. Cada vez que realicemos un movimiento de búsqueda habremos de comparar dos cadenas por si coinciden: la que buscamos y la que se encuentra en la actual posición en la tabla.

Esta comparación de series se presta obviamente a ser realizada mediante una subrutina, porque no sólo habrá de usarse más de una vez en el programa, sino que nos permitirá dividir el pro-

Pila saltarina

Toda llamada a una subrutina implica un "salto de ida y vuelta", posibilitado mediante la conservación del valor del contador de programa. Éste se sustituye por la dirección de llamada de la subrutina, y al finalizar ésta se le restituye su valor anterior. La pila es un área de la memoria empleada por el procesador con el fin de guardar allí el valor del contador, es decir la dirección de retorno, y el índice de pila es un registro de 16 bits de la CPU que contiene siempre la dirección del siguiente byte libre en el espacio de la pila.
Cuando se encuentra un JSR en, por ejemplo, la dirección $COBA, la CPU automáticamente sitúa $COBD (que es la dirección de la instrucción que sigue a la JSR) en el contador del programa. Al momento de ejecutar la instrucción JSR, el contenido del contador del programa es colocado (*push*) en la pila por la CPU y sustituido por $F000. Comienza así la ejecución de la subrutina que se dará por finalizada al encontrar RTS (ReTurn from Subroutine: vuelta de la subrutina), la cual obliga a extraer (*pull, pop*) la dirección de vuelta, el valor $COBD, de la pila y restituirlo al contador del programa



Kevin Jones

blema en dos útiles secciones. Además de ser una subrutina digna de tenerla disponible para otros programas.

La subrutina necesita dos datos suministrados por el programa que la llama: las direcciones de las dos cadenas a comparar. Puesto que la subrutina tiene que repasar las cadenas byte a byte, será mejor trasladar estos valores a los registros X e Y, donde serán necesarios. Además, la subrutina ha de retornar dos valores, uno que indique si hubo o no coincidencia, y otro que diga la dirección de la pareja encontrada.

Verdadero o falso

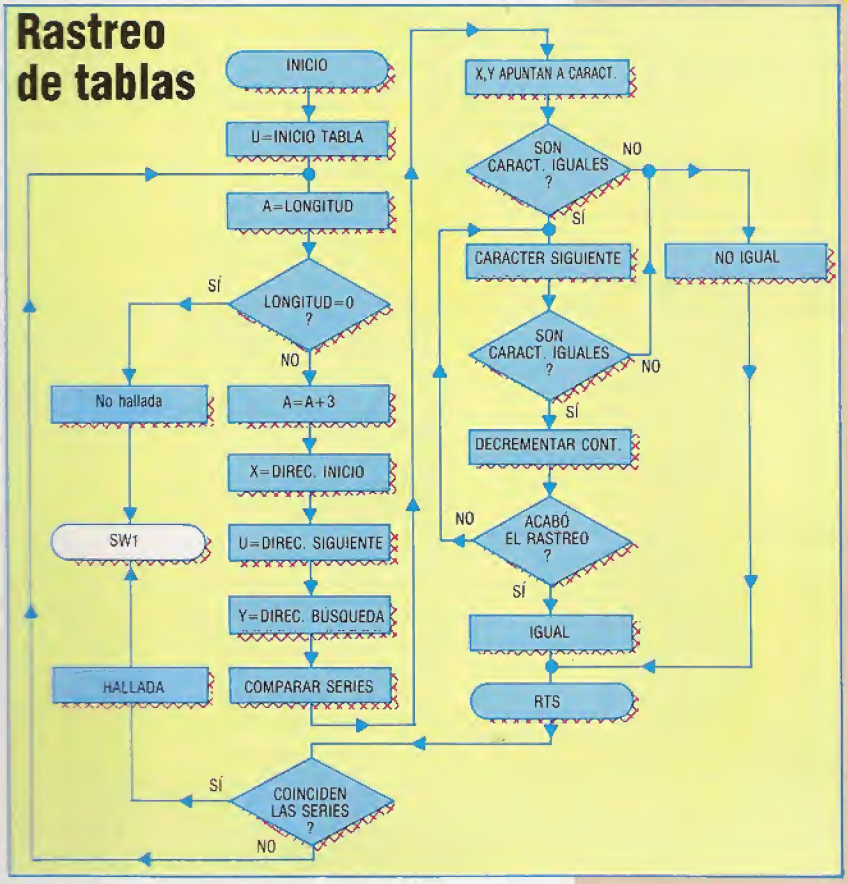
Es posible pasar un parámetro booleano (verdadero o falso) mediante los flags del registro de código de condición, pero es preciso tener puntual conocimiento del efecto de cada instrucción sobre los flags. En nuestro programa los valores que llevaremos a la rutina principal serán o todos ceros ($00), o todos unos ($FF), según que hubo coincidencia o no, respectivamente.

Para generalizar la subrutina, no suministraremos la dirección concreta de la pareja hallada, sino que dejaremos que el registro quede señalando la dirección donde se encuentra la dirección de la pareja hallada. Lo cual tiene una ventaja adicional, y es que el registro X, al recorrer la cadena byte a byte, acabará conteniendo esta información de manera automática.

Una última observación: nuestro programa contiene una nueva instrucción del 6809. Es TST (TeST: comprobación), la cual no afecta a registro alguno y se limita a activar los flags según el valor actual del registro nombrado.

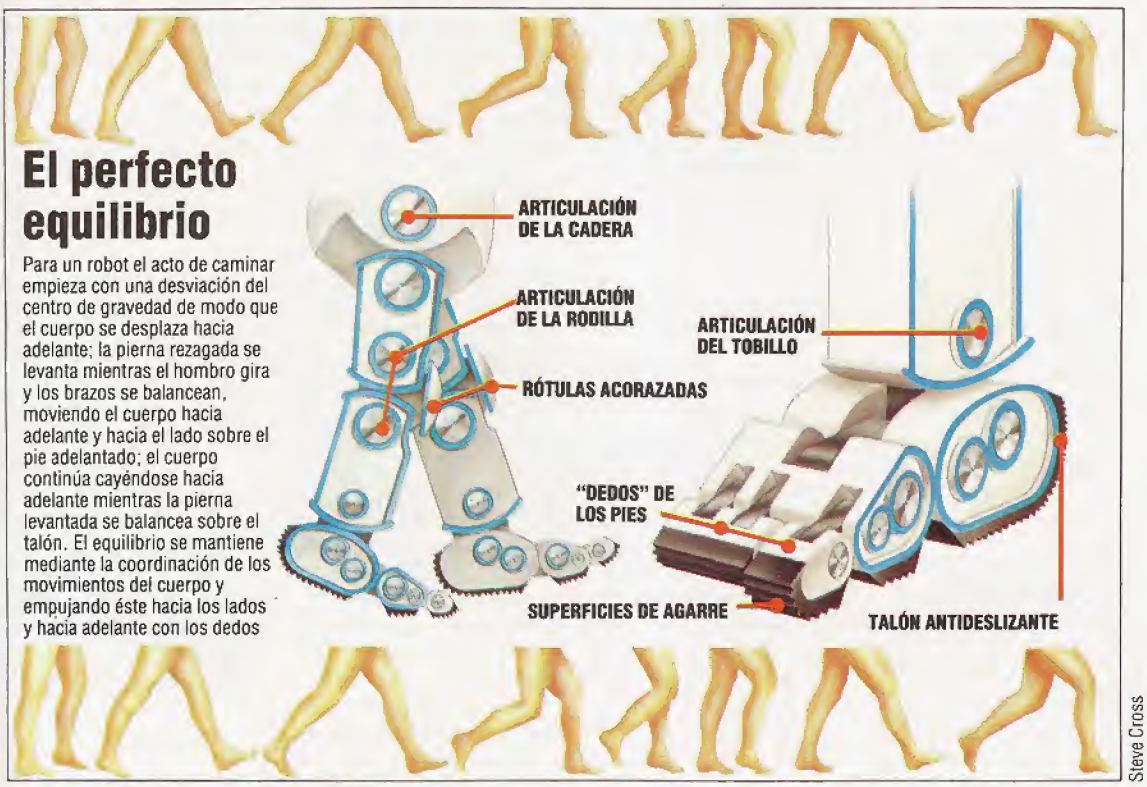
```
TABLA   EQU   $10
STRNG   EQU   $12
ADDRS   EQU   $14
        ORG   $1000      Comienzo programa principal
        LDU   TABLA      Comienzo de la tabla en U
BUCLE1  LDA   ,U         Toma el byte de longitud
        BEQ   NTFND1     Si cero, ir fin de tabla
        ADDA  #3         Suma uno por byte longitud,
        TFR   U,X        dos por dirección longitud
        LEAU  A,U        cadena, dando la longitud de
        LDY   STRNG      esta entrada en la tabla
        BSR   COMPAR     Coloca el inicio de la serie
        TSTA             en X
        BEQ   FOUND1     Hace que U apunte a la
        BRA   BUCLE1     siguiente entrada
FOUND1  LDD   ,X         Y apunta al comienzo de la
        BRA   FINAL1     serie de rastreo
NTFND1  LDD   #0         Compara las dos series
FINAL1  STD   ADDRS      Mira si son iguales
        SWI              Si lo son, entonces GOTO
COMPAR  LDB   ,X+        FOUND1
        CMPB  ,Y+        Si no lo son, toma la siguiente
        BNE   NOTEQ      entrada de la tabla
BUCLE2  LDA   ,X+        Si es hallada, X apuntará a la
        CMPA  ,Y+        dirección que precisamos
        BNE   NOTEQ      Si no es hallada, la dirección
        DECB             será cero
        BGT   BUCLE2     Guarda lo que encontramos
        CLRA             Fin del programa principal
        BRA   FINAL2     Inicio de la subrutina
NOTEQ   LDA   #$FF       Toma los bytes de longitud y
FINAL2  RTS              apunta X e Y a los primeros
        END              caracteres
                         Si las series no tienen igual
                         longitud, GOTO NOTEQ
                         Toma el siguiente carácter de
                         la tabla de series
                         Compara éste con el siguiente
                         de la serie de rastreo
                         Se detiene si no son idénticos
                         Si lo son, toma uno del
                         indicador de posición
                         Toma el carácter siguiente
                         Pone A a cero para indicar que
                         las series son idénticas
                         Lo pone a uno si no son
                         idénticas
                         Vuelta al programa principal
```

Rastreo de tablas



¡En marcha!

En esta ocasión analizaremos los tres métodos principales de movimiento del robot y el medio más eficaz de controlarlo



Steve Cross

Mucho antes de que un niño aprenda a caminar, puede coger objetos y demostrar su inteligencia de numerosas maneras; pero caminar es una habilidad cuyo desarrollo lleva mucho tiempo y exige una considerable práctica antes de que adquiera un carácter automático.

Se puede hacer que los robots "caminen", pero las técnicas implicadas son muy distintas de los métodos que emplean los seres humanos. El robot puede tener piernas, capaces de balancearse hacia adelante y hacia atrás en una aproximación al caminar humano, pero cada una de ellas está provista de un pie que posee ruedas en su base. Estas ruedas están equipadas con mecanismos para impedir el movimiento de retroceso. De modo que un robot de este tipo sigue, cuando "camina", una secuencia de acciones preestablecida. La desventaja de este procedimiento es que es difícil desarrollar una forma de gobernar el robot; éste tenderá a moverse sólo hacia adelante y, además, sus movimientos serán imprecisos.

Una solución mucho mejor sería hacer que los robots caminaran levantando primero una pierna y luego la otra, como hacen los humanos, en vez de balancear simplemente cada extremidad a través de un arco limitado. El principal problema de este enfoque es que el robot debe ser capaz de mantenerse en equilibrio sobre una sola pierna mientras va caminando. Se han probado diversas soluciones: éstas incluyen inclinar el cuerpo del robot hacia los lados sobre un riel de modo que el centro de gravedad del robot se sitúe directamente encima de la pierna que está soportando su peso. Si se desarrollara un sistema de este tipo, los robots podrían caminar de una forma eficaz. En teoría, se podría diseñar un robot que subiera las escaleras y le llevara a su dueño el desayuno por la mañana. Pero en la práctica, a pesar de que un robot que suba las escaleras es bastante factible, ya sería más difícil desarrollar un robot que "supiera" cuándo terminaba la escalera, debido a los aparatos adicionales necesarios para detectar el último escalón.

Un enfoque alternativo ha sido el de montar robots sobre cadenas similares a las de los tanques. Este sistema ofrece la ventaja de permitir que el robot se mueva sobre terreno irregular. Los ejércitos más modernos utilizan este tipo de robots oruga para llevar a cabo peligrosos trabajos de desactivación de bombas; estas máquinas pueden maniobrar a través de escombros y son capaces de cubrir un terreno moderadamente desnivelado.

Las cadenas son sólidas y, a la vez, fáciles de dirigir, pero poseen dos inconvenientes fundamentales. El primero es que, dado que la mayoría de los robots son bastante pequeños, también son reducidas las dimensiones de las cadenas y, en consecuencia, no se pueden salvar grandes obstáculos. Un tanque de combate de tamaño natural puede abrirse paso prácticamente ante cualquier obstáculo con facilidad; pero esto se debe sólo a que los tanques son grandes, pesados y potentes. Si un carro de combate tratara de pasar sobre un objeto tan gran-

de que su centro de gravedad se desplazara fuera de la zona de sus cadenas, se caería. Esto de hecho sucede a veces cuando el terreno es demasiado irregular. Lo mismo le sucedería a un robot oruga si intentara subir por superficies demasiado escarpadas.

La segunda desventaja es que las cadenas no se pueden controlar con precisión. El mecanismo de dirección consiste en detener una cadena de modo que la otra continúe en movimiento; el robot (o el tanque) se desplaza, por consiguiente, describiendo un arco. Cuando esto sucede, la banda estacionaria puede que se desplace ligeramente y la posición final no será la esperada. Un carro de combate conducido por una persona puede corregir rápidamente cualquier error de este tipo; pero para un robot las necesarias correcciones del recorrido son considerablemente más complejas.

Para el control del robot, obviamente es deseable contar con un juego de instrucciones que siempre hagan que el robot se desplace exactamente hasta el lugar correcto, encaminándolo en una dirección predecible con exactitud. Precisamente por este motivo la forma más común de movimiento de los robots es mediante ruedas. Éstas poseen varias ventajas evidentes, al ser simples, eficaces y capaces de producir un movimiento mucho más uniforme que el que jamás se podría obtener con las piernas.

Una vez aceptado que el robot ha de tener ruedas, el único problema es el control exacto del movimiento. Consideremos, por ejemplo, un automóvil de juguete accionado por cuerda. Éste va sobre ruedas pero no es un robot, dado que no posee ningún medio que le permita "conocer" su posición en cualquier momento dado. Lo que se necesita es un sistema de coordenadas que se pueda utilizar para determinar la posición de un objeto sobre una superficie; el sistema más común a este fin se vale de coordenadas cartesianas. Con este sistema es posible determinar la posición exacta de un robot y especificar los movimientos que se precisan para desplazarlo hasta otra posición definida. Todo lo que se necesita luego es un dispositivo para asegurar que el robot se pueda mover con precisión dentro de este marco de referencia.

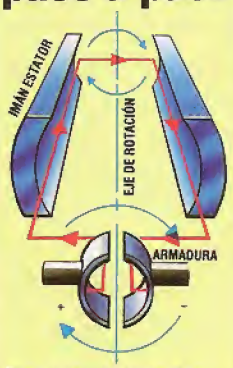
Si bien ocasionalmente se utiliza la energía hidráulica o neumática, el método más común para desplazar robots móviles es mediante un motor eléctrico. Como hemos visto en nuestro apartado de *Bricolaje* (pp. 1065 y 1092), un motor eléctrico sencillo puede proporcionar movimiento y un discreto control sobre la dirección. El mismo no es apto para un control de precisión; un motor eléctrico sencillo siempre efectúa un giro de al menos 180° antes de un detención, y con frecuencia la inercia hace que rote un poco más que eso.

Por consiguiente, para el control de robots normalmente se utiliza un motor paso a paso. Éste contiene una gran cantidad de bobinas y, si bien los diseños pueden variar enormemente, el principio general del motor paso a paso permite especificar porciones de rotación pequeñas y exactas, con poco error por exceso (rotar más de lo que debería) o por defecto (rotar demasiado poco).

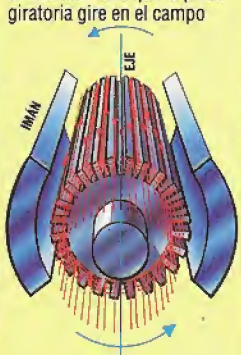
Los robots que emplean motores paso a paso están ampliamente difundidos. Tales robots suelen tener acoplado un lápiz que les permite trazar una línea sobre la superficie sobre la cual se están desplazando. Estos robots "esgrimidores" de lápices se llaman *tortugas* y se dice que los diseños que producen son *gráficos tortuga*. Están capacitados para efectuar un movimiento preciso; su precisión se puede juzgar instruyéndolos para que dibujen una forma cerrada, como un rectángulo o una estrella, comprobando si la línea trazada se cierra o no sobre sí misma en el punto de comienzo.

Los motores paso a paso y las coordenadas cartesianas, por consiguiente, nos pueden proporcionar un método relativamente preciso de controlar el movimiento de un robot. No obstante, si el robot ha de hacer algo más que simplemente dar vueltas por encima de una superficie dada, tropezando con objetos, necesitará ser capaz de responder con rapidez y precisión a las condiciones externas. En el próximo capítulo nos centraremos en ello.

Medidas paso a paso



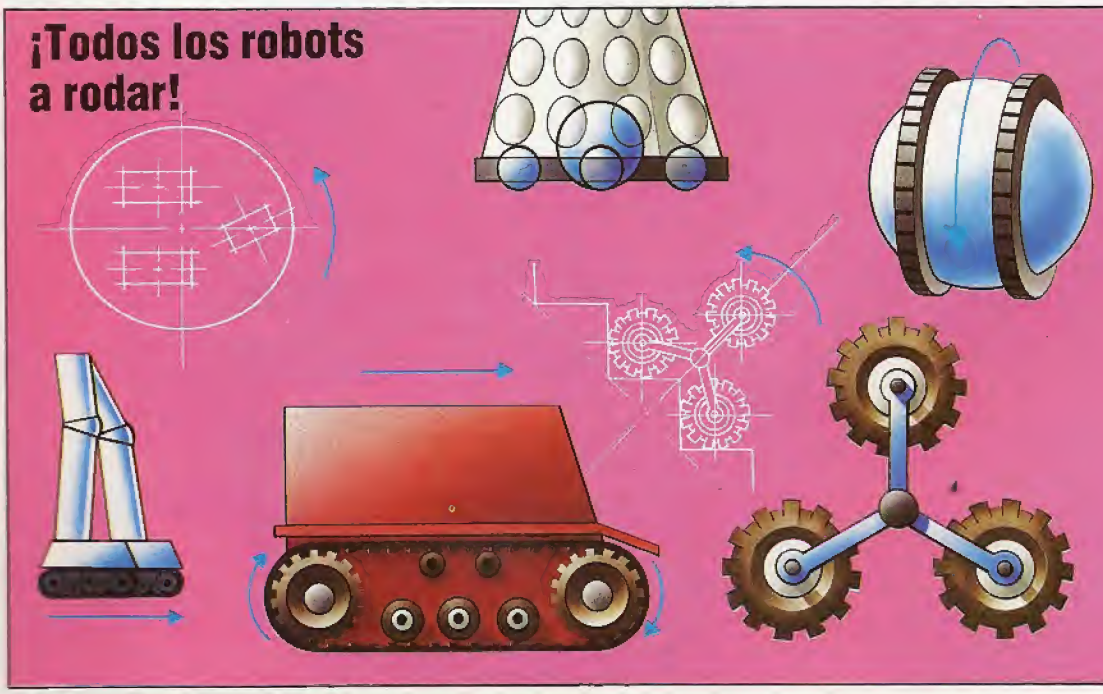
En el motor eléctrico más simple, un flujo de corriente en la bobina giratoria crea un flujo magnético contrario al del campo magnético del imán estator; esta oposición de fuerzas hace que la pieza giratoria gire en el campo



El motor paso a paso posee muchas bobinas, a veces centenares. La conmutación de corriente de una bobina a otra hace que el conjunto gire en incrementos de arco controlables exactamente

Kevin Jones

¡Todos los robots a rodar!



He aquí algunos de los posibles escenarios para el desplazamiento de un robot. Las piernas con cadenas o correas ofrecen agarre sacrificando maniobrabilidad, pero permiten una marcha uniforme sin elevación de piernas, minimizando, por consiguiente, los problemas relativos al equilibrio. Los vehículos robot con este sistema son muy corrientes en los equipos de exploración planetaria y en las patrullas desactivadoras de bombas.
El formato de tres ejes es la única adaptación que permite al robot subir escalones.
Una gran rueda que gire rodeada de estabilizadores es muy fácil de gobernar, pero es sensible a las superficies irregulares. La disposición de dos ruedas fijas y una dirigida es el mínimo necesario para la estabilidad.
Distribuir la carga sobre ruedas motrices es acertado, pero eleva el centro de gravedad y reduce la estabilidad

Kevin Jones

Divide y vencerás

En este capítulo de nuestro curso de LOGO crearemos unos interesantes gráficos basados en la utilización de los procedimientos denominados "recursivos".

Nuestro primer programa está diseñado para dibujar formas de "árbol". Para comenzar, podemos simplemente dibujar un tronco con una rama a la izquierda y una rama a la derecha. Luego éstas se pueden formar exactamente de la misma manera (si bien serán más pequeñas), con una rama principal central y ramitas a la derecha y a la izquierda. Si se continuara este proceso se crearía gradualmente una forma de árbol.

Éste es un buen ejemplo de cómo se puede utilizar la recursión en LOGO.

Nuestro procedimiento para dibujar este árbol "binario" exige dos entradas: una para la longitud del tronco y la otra para el número de "nivel". La longitud de las ramas se reduce a la mitad a cada nivel, alejándose del tronco.

```
TO RAMA :LENGTH :NIVEL
  IF :NIVEL=0 THEN STOP
  FD :LENGTH
  LT 45
  RAMA (:LENGTH/2)(:NIVEL-1)
  RT 90
  RAMA (:LENGTH/2)(:NIVEL-1)
  LT 45
  BK :LENGTH
END
```

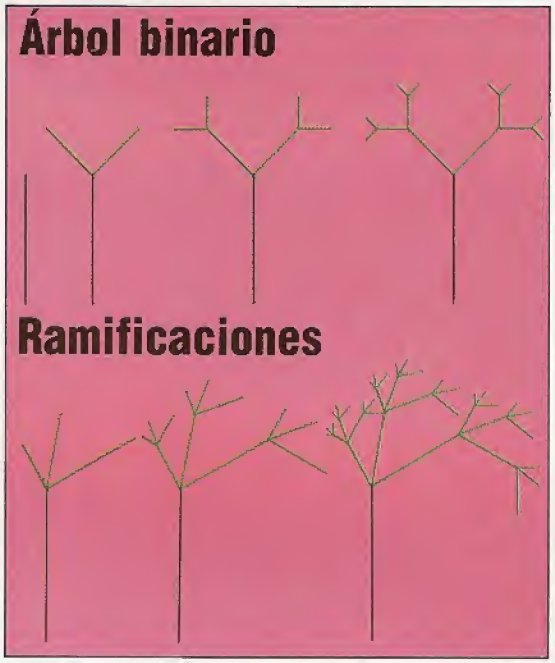
Observe que el procedimiento es transparente. Esto es importante, puesto que, de lo contrario, el "estado" de la tortuga (su posición y encabezamiento) cambiaría cada vez que el procedimiento se llamara a sí mismo, haciendo imposible continuar con el dibujo.

Hay que admitir que este procedimiento no produce un árbol realista; para hacerlo más interesante, se puede modificar el procedimiento de diversas maneras.

Ahora proponemos una versión que dibuja tres ramas, cada una de ellas de diferente longitud, en cada nivel:

```
TO RAMA1:LENGTH :NIVEL
  IF :NIVEL=0 THEN STOP
  FD :LENGTH
  LT 30
  RAMA1(:LENGTH/3)(:NIVEL-1)
  RT 40
  RAMA1(:LENGTH/2)(:NIVEL-1)
  RT 50
  RAMA1 (:LENGTH/1.5)(:NIVEL-1)
  LT 60
  BK :LENGTH
END
```

Pruebe otras modificaciones para producir árboles más parecidos a la realidad.



Polígonos a cuadros

El siguiente procedimiento dibuja un cuadrado, lo divide en cuatro, luego divide cada parte en cuatro, y así sucesivamente:

```
TO TABLERO :LENGTH :NIVEL
  IF :NIVEL=0 THEN REPEAT 4 [FD :LENGTH RT 90]
      STOP
  TABLERO (:LENGTH/2)(:NIVEL-1)
  FD (:LENGTH/2)
  TABLERO (:LENGTH/2)(:NIVEL-1)
  RT 90
  FD (:LENGTH/2)
  LT 90
  TABLERO (:LENGTH/2)(:NIVEL-1)
  BK (:LENGTH/2)
  TABLERO (:LENGTH/2)(:NIVEL-1)
  LT 90
  FD (:LENGTH/2)
  RT 90
END
```

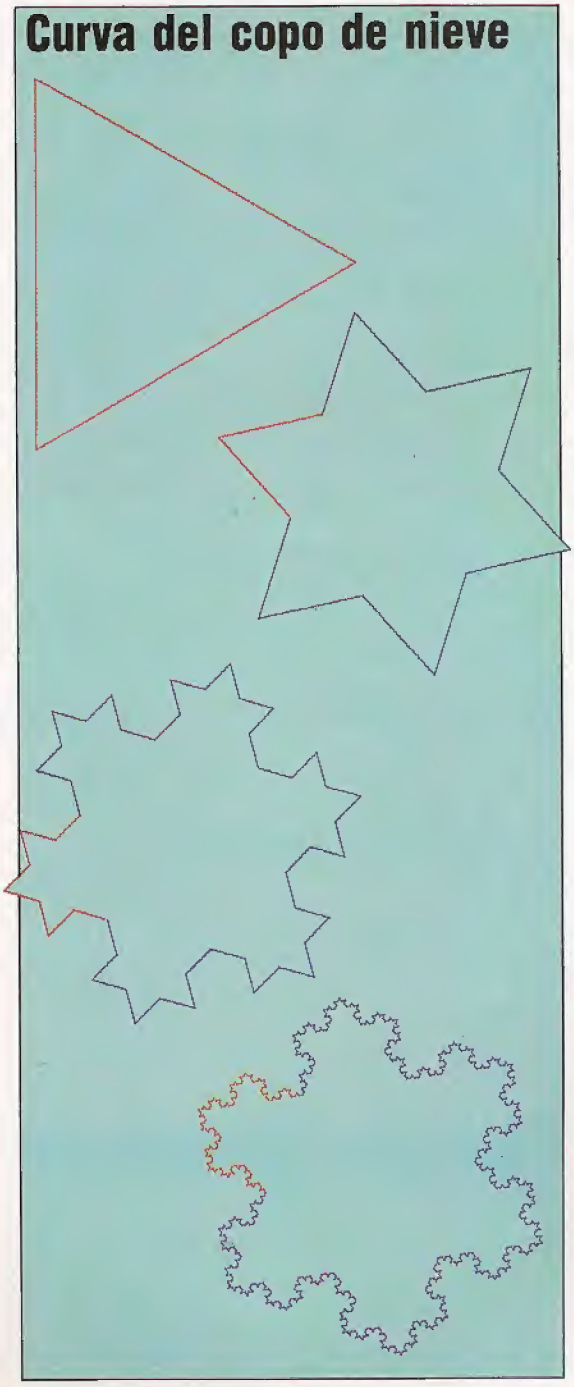
Escriba un procedimiento similar que divida un triángulo en cuatro más pequeños, que luego divida éstos en otros cuatro, y así sucesivamente.

Copos de nieve

Dibuje primero un triángulo equilátero. Divida cada lado en tres partes iguales y dibuje un nuevo triángulo equilátero en la sección central. Borre las líneas compartidas, luego repita esta secuencia para

cada lado de la nueva forma, y continúe el proceso. Se dice que la forma resultante es la "curva del copo de nieve", debido a su aspecto.

```
TO NIEVE :TAMAÑO :NIVEL
  REPEAT 3 [LADO :TAMAÑO :NIVEL RT 120]
END

TO LADO :TAMAÑO :NIVEL
  IF :NIVEL=0 THEN FD :TAMAÑO STOP
  LADO (:TAMAÑO/3)(:NIVEL-1)
  LT 60
  LADO (:TAMAÑO/3)(:NIVEL-1)
  RT 120
  LADO (:TAMAÑO/3):(NIVEL-1)
  LT 60
  LADO(:TAMAÑO/3)(:NIVEL-1)
END
```



Observe que LADO no es transparente, sino que, en cambio, se ha construido como para dejar a la tortuga en el lugar adecuado para dibujar el lado siguiente.

Si este proceso de división se continuara indefinidamente (los matemáticos utilizan la frase "hasta el límite"), el resultado sería una curva que posee una longitud infinita y que, sin embargo, ¡delimita una superficie finita (fija)! Se puede demostrar que esta curva no es ni unidimensional ni bidimensional, sino que está más bien en un punto intermedio entre ambos extremos.

Se podría construir una curva similar comenzando con un cuadrado, dividiendo cada lado en tres partes iguales, construyendo cuadrados en las secciones del medio, y así sucesivamente. Trate de escribir un procedimiento que haga esto.

La serie de curvas que podemos observar en la página contigua la inventó un matemático llamado Sierpinski y sirven para rellenar espacios. Si el proceso se continúa hasta el límite, el resultado es una curva (una línea unidimensional) que pasa a través de cada uno de los puntos del cuadrado que la rodea (una forma bidimensional). Existen muchas otras "curvas para rellenar espacios" que tienen este curioso comportamiento.

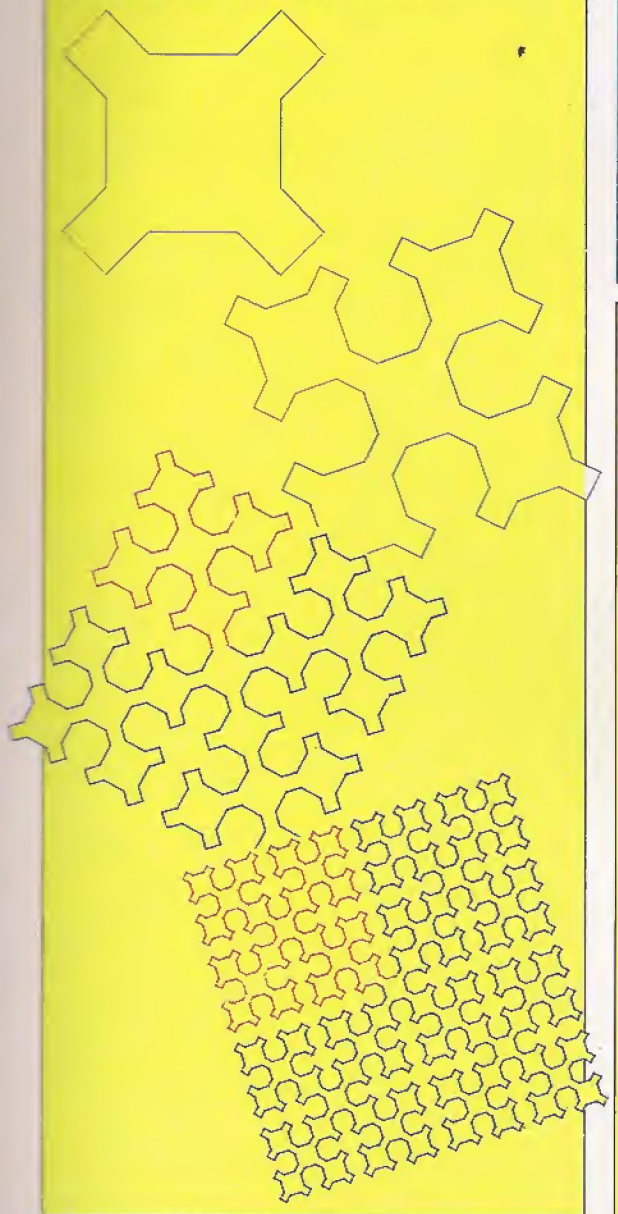
El procedimiento empleado para dibujar esta curva es bastante complicado. La curva del nivel 1 se compone de cuatro lados (coloreados en azul) que se unen mediante cuatro diagonales (coloreadas de rojo). De modo que el procedimiento principal, SIERP, no hace más que dividir el proceso en cuatro secciones para que el procedimiento UN.LADO las trate de una en una.

Consideremos sólo uno de los lados. Éste se compone de tres líneas: una diagonal, otra horizontal o vertical y una tercera diagonal. En el nivel 2, cada diagonal se sustituye por otro conjunto, más pequeño, de tres líneas, y la línea horizontal o vertical se reemplaza por dos conjuntos similares de tres líneas unidos mediante una línea. Para pasar de un nivel a otro se lleva a cabo el mismo proceso.

He aquí los procedimientos para dibujar las curvas. Observe cómo se utiliza la instrucción del LOGO MAKE para inicializar DIAG:

```
TO SIERP :LADO :NIVEL
  MAKE "DIAG :LADO / SQRT (2)
  REPEAT 4 [UN.LADO :NIVEL RT 45 FD :DIAG RT 45]
END

TO UN.LADO :NIVEL
  IF :NIVEL=0 STOP
UN.LADO (:NIVEL-1)
  RT 45
  FD :DIAG
  RT 45
  UN.LADO(:NIVEL-1)
  LT 90
  FD :LADO
  LT 90
  FD :LADO
  LT 90
  UN.LADO(:NIVEL-1)
  RT 45
  FD :DIAG
  RT 45
  UN.LADO(:NIVEL-1)
END
```

Curva de Sierpinski



Complementos al LOGO

Las versiones LCSI de LOGO utilizan SETPOS para establecer la posición de la tortuga. Ésta requiere una lista como entrada, de modo que las dos coordenadas se deben combinar con LIST. P. ej.:

```
SETPOS LIST 45 67
```

Recuerde, asimismo, que las versiones LCSI emplean una sintaxis diferente con IF. Una regla de detención típica sería:

```
IF NIVEL=0 [STOP]
```

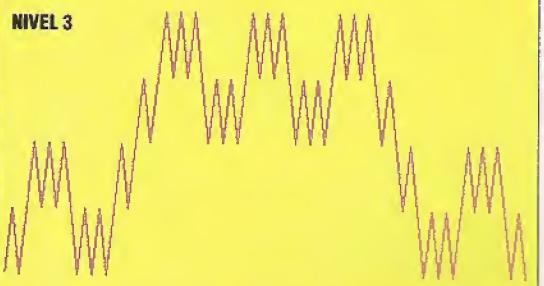
Procedimientos (4)

Nuestra ilustración muestra una serie de formas que en su límite definen una curva que no tiene gradiente en ningún punto. El primer nivel se compone de dos líneas: una que va hacia arriba, la otra hacia abajo. Para pasar al siguiente nivel, sustituimos la línea ascendente por una línea quebrada con seis partes. Ésta se eleva hasta la mitad de la altura de la línea original, luego desciende; vuelve a elevarse hasta la mitad, sigue hasta la altura total, cae hasta la media altura y sube hasta la altura total. La descendente se divide en seis secciones.

NIVEL 1

NIVEL 2

NIVEL 3



Intente escribir un conjunto de procedimientos que dibuje esta serie de curvas. Deberá utilizar SETXY en lugar de FD y RT. Su procedimiento de nivel superior habrá de dividir la tarea en dos partes: una para ascender, la otra para descender. Luego tendrá que escribir dos procedimientos separados para tratar con las dos partes del procedimiento. Estos procedimientos se pueden llamar entre sí y a sí mismos.

Respuestas a los ejercicios

Procedimiento que dibuja una torre de cuadrados:

```
TO TORRE :TAMAÑO
  IF:TAMAÑO<5 THEN STOP
  CUADRADO :TAMAÑO
  MOVER :TAMAÑO
  TORRE (:TAMAÑO/2)
END

TO CUADRADO :TAMAÑO
  REPEAT 4 [FD :TAMAÑO RT 90]
END

TO MOVER :TAMAÑO
  FD :TAMAÑO
  RT 90
  FD (:TAMAÑO/4)
  LT 90
END
```

La suma de las partes

Iniciamos una serie en la que analizaremos paquetes de software integrado. Veamos, en primer lugar, qué significa este término

La integración representa una de las tendencias más interesantes que ha habido en el software. Por el momento se aplica principalmente a los sistemas de gestión, aunque ya sus técnicas se han empezado a introducir en los micros personales. Un ejemplo de ello es el Sinclair QL, cuyos cuatro paquetes de software incorporan los principios básicos de la integración (véanse pp. 982-983).

El logro fundamental de la integración es el de permitir que el programador pase de un paquete a otro diferente de manera rápida y fácil. En un sistema ideal no se tendría que salir de un programa, regresar al sistema operativo, intercambiar discos y después comenzar con otro programa. Para que sea eficaz, el cambio de aplicación ha de concretarse casi en pulsar sólo una tecla, y algunos programas, como el *Lotus 1-2-3* y el *Framework*, de Ashton Tate, lo consiguen.

También resulta útil poder transferir datos fácilmente entre paquetes. Por ejemplo, se podría crear una columna de cifras de ventas anuales para su negocio con el programa de hoja electrónica, luego transferir entera esa columna al programa para tratamiento de textos, con el que escribiría el informe anual. Se podrían utilizar los nombres y las direcciones de un archivo de base de datos con el procesador de textos para escribirles una carta personalizada a todas las personas del archivo. En el Lisa y el Macintosh esta facilidad se amplía hasta el punto de que uno puede crear un dibujo a mano alzada con el programa para gráficos y después llevarlo directamente a un documento elaborado con tratamiento de textos.

Además, todos los distintos programas deberían trabajar de la misma forma y producir similar sensación al ser utilizados. Los trazados de pantalla, las teclas de instrucciones, los avisos, los mensajes de error —todos los aspectos de la "interface para el usuario"— deberían ser idénticos o comparables. De no ser así, el usuario no puede pasar con entera confianza de una zona a otra sin tener que detenerse y adaptarse a los cambios en los procedimientos de operatoria. Ello supone una interrupción en la fluidez con que se puede utilizar el software e impide que se lo pueda explotar al máximo.

Un efecto secundario inmediato es que el paquete se vuelve más fácil de aprender. Tener que aprenderse cinco nuevos programas de aplicaciones (algunos activados por menú, otros activados por comandos, todos con distintos formatos de comandos) es una tarea que puede resultarle intimidante a cualquiera. Pero si todos funcionan de la misma manera, el usuario sólo necesita aprenderse uno. Esta característica se conoce como *uniformidad* y se suele citar a menudo cuando se habla de software integrado.

Hemos concluido, entonces, que el software integrado implica tres principios de diseño: facilidad para pasar de una aplicación a otra, libertad para intercambiar datos entre aplicaciones y uniformidad de formato. Estas características contribuyen a hacer que el ordenador sea más accesible al usuario medio, cuyas necesidades se pueden satisfacer con dos o tres aplicaciones de software. Sin duda, contribuirá asimismo a aumentar la popularidad del ordenador personal, dado que su utilización resultará más eficaz y sencilla.

No obstante, el software integrado tiene algunos inconvenientes. El más importante de ellos es el hecho de que los paquetes de software integrado necesitan grandes cantidades de RAM para operar. Imagine el lector tratar de colocar un procesador de textos, una hoja electrónica y una base de datos (las tres aplicaciones que se integran con mayor asiduidad) en 16 o 32 Kbytes. Probablemente se pueda hacer, pero no quedará mucho espacio (en caso de que quede) para almacenar datos. Es esta circunstancia la que limita el software integrado a máquinas con una gran memoria: en general, a ordenadores con 128 Kbytes o más. Por supuesto, los programas que están integrados pueden compartir algunas rutinas, de modo que sólo es necesario escribir una vez las operaciones de almacenamiento en disco y otras tareas del sistema. Sin embargo, cada aplicación tiene sus propias exigencias especiales, y éstas ocupan espacio en RAM.

Una segunda desventaja del software integrado es consecuencia del mismo problema del almacenamiento. Para hacer un ahorro en la cantidad de memoria que requiere un programa, los productores de software recurren a las aplicaciones individuales. Un paquete de proceso de textos que esté incorpo-

Para revisar...
Entre los paquetes integrados que estudiaremos en esta serie se incluirán el *Lotus 1-2-3, Open access, Symphony* y *Framework*

Cortesía de Microscope
Ian McKinnell

rado en un paquete integrado con otras dos o tres aplicaciones no puede ser tan eficaz y completo como un paquete exclusivo. El principal motivo reside en que un programa unitario puede ocupar tanta memoria él solo como todos los programas integrados juntos.

Podemos encontrar un ejemplo en dos programas que se ejecutan en el IBM PC y ordenadores similares. *Multimate* es un programa diseñado alrededor del software que se utiliza en el equipo de proceso de textos exclusivo Wang. Posee muchas opciones para crear y formatear texto de las que no disponen otros programas más pequeños, lo que simplifica de manera considerable la creación de documentos muy extensos y complicados. El *Multimate* requiere, él solo, 192 Kbytes de RAM para operar. El *Lotus 1-2-3*, un programa integrado de tratamiento de textos, hoja electrónica y base de datos, también exige 192 Kbytes. Pero el mismo espacio que utiliza *Multimate* para una aplicación debe ahora contener todo el software necesario para tres programas completos en el *Lotus*. Como resultado, en este último el tratamiento de textos se limita a una simple escritura de informes.

Una tercera desventaja de la integración también se deriva de una de sus ventajas. Es importante, como ya hemos visto, que los programas integrados sean parecidos, de modo que sean fáciles de aprender y de manejar. Lamentablemente, los escritores de software tienen que hacer ciertas concesiones para que ello sea posible. Puede ser que la mejor forma de hacer funcionar una hoja electrónica no sea la más adecuada para hacer operar una base de datos o un procesador de textos, de modo que se tiende a combinar los elementos óptimos de diseño para cada uno en una mezcla utilizable. Microsoft se encontró con este problema cuando diseñó un procesador de textos individual, el Microsoft Word. La empresa deseaba que la visualización en pantalla y la operación del programa fueran compatibles con su hoja electrónica, el Multiplan, que tan enorme éxito había obtenido, para que fuera sencillo integrar los dos programas. Microsoft incluyó en el Word el mismo menú en pantalla que a los usuarios del Multiplan les resultaba tan útil, sólo para descubrir que a los escritores que necesitaban un programa como el Word no les gustaba tener un menú en la pantalla todo el tiempo.

Lo que hay que recordar con respecto al software, es que éste tiene que hacer lo que el usuario desea. Si una persona debe realizar varias tareas, como escribir cartas, llevar cuentas sencillas y tener listas de correspondencia, el software integrado puede hacer que el trabajo sea mucho más fácil. Pero hay que hacer sacrificios, y alguien que desee escribir una novela o informes de empresa muy extensos en su microordenador, quizá tenga que seguir utilizando programas individuales, separados, de tratamiento de textos, hojas electrónicas y bases de datos. No obstante, a medida que los escritores de software vayan aprendiendo más sobre la compresión de instrucciones de ordenador en espacios de memoria cada vez más pequeños, y a medida que la memoria de algunas máquinas personales comience a crecer, el software integrado irá adquiriendo una importancia cada vez mayor tanto para el usuario personal como para el usuario de gestión. A modo de indicio de lo que será el futuro, ya hay dos ordenadores que se están vendiendo con software integrado: el QL de Sinclair y el Plus/4 de Commodore.

En futuros capítulos de esta serie analizaremos algunos de los programas integrados que están teniendo un mayor efecto en el desarrollo de software. Examinaremos dos enfoques distintos a la integración: el que ejemplifican el *Lotus 1-2-3* y el que representan el Lisa y el Macintosh.

Las reglas del juego

Configuración completa
La integración es el principio operativo del Macintosh de Apple, en el cual se produjeron estas ilustraciones. Multiplan (la hoja electrónica), MacWrite (el procesador de textos) y MacPaint (el programa para gráficos) se comunican directamente a través del sistema operativo, de modo que las tres aplicaciones se convierten en una

Código común
Estas visualizaciones de hoja electrónica y tratamiento de textos muestran claramente la uniformidad de formato entre aplicaciones integradas

Cargos transferidos
Los datos se han transferido desde la hoja electrónica al procesador, demostrando la importancia de la compatibilidad y transferencia en el software integrado

Ian McKinnell

Búsqueda de datos

En esta oportunidad consideraremos un procedimiento para buscar un dato específico en una lista de valores

Si en capítulos anteriores hemos mostrado diferentes maneras (burbuja, máximo) de clasificar datos, ya fueran numéricos o alfanuméricos, ahora vamos a mostrar cómo buscar un dato determinado en una lista de valores. Así, contamos con la llamada *búsqueda secuencial*, basada en la comparación de un dato determinado con los contenidos en una tabla, uno por uno. Este proceso puede terminar con dos posibles finales lógicos, a saber: que el dato buscado no se halla en la tabla, y que el dato buscado se encuentra en la tabla y la posición que ocupa.

Presentamos dos modos diferentes de realizar dicha búsqueda; la figura 1 lo hace mediante el uso de contadores en forma tradicional, mientras que la figura 2 lo hace empleando FOR...NEXT.

```
10 REM METODO DE BUSQUEDA SECUENCIAL
20 INPUT "NUMERO DE ELEMENTOS"; N
30 DIM T(N)
40 FOR M=1 TO N
50 INPUT T(M)
60 NEXT M
70 INPUT "VALOR A BUSCAR"; R
80 I=1
90 IF I>N THEN PRINT "VALOR NO
   ENCONTRADO" : END
100 IF R=T(I) THEN PRINT "EL VALOR OCUPA LA
   POSICION";I : END
110 I=I+1
120 GOTO 90
```

Figura 1

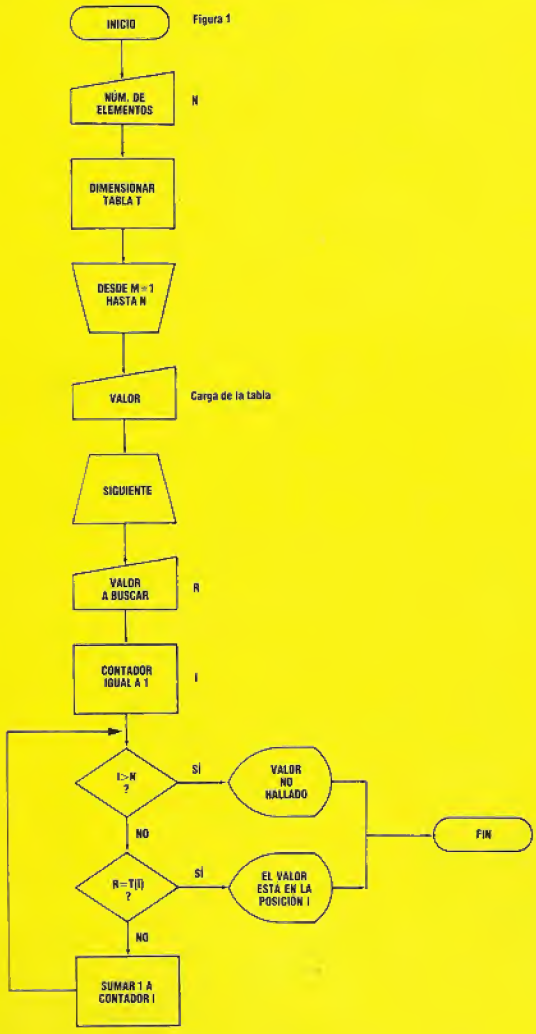
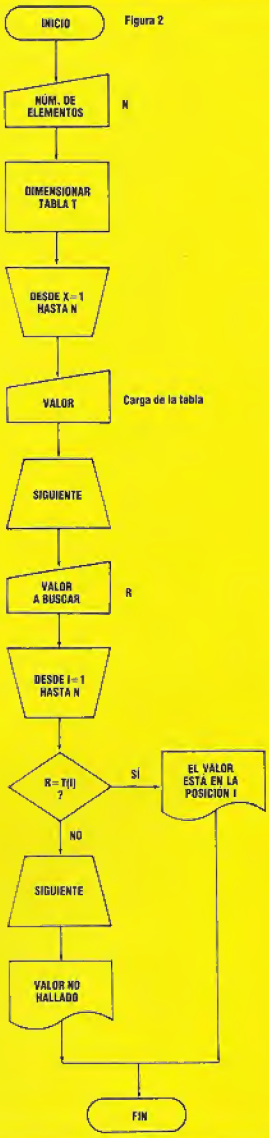


Figura 2



Juguete para artistas

Koala-pad proporciona una forma sencilla de producir gráficos de alta calidad en el Commodore 64

A pesar de su capacidad para producir excelentes efectos gráficos, hasta el momento no se ha dotado al Commodore 64 de los periféricos para gráficos de alta calidad que se han producido para el BBC Micro, por ejemplo. Esto probablemente es consecuencia de las dificultades asociadas con la producción de gráficos en alta resolución en el 64, problemas que han disuadido a los fabricantes de producir tal dispositivo. Ahora, sin embargo, Audiogenic ha empezado a importar un periférico para gráficos fabricado por una empresa norteamericana, Koala Technology, que permite a los usuarios del Commodore 64 un acceso más sencillo a las capacidades para gráficos en alta resolución de la máquina.

A diferencia de tablillas al tacto similares, como la Grafpad (véase p. 649), la Koala-pad es ligera y compacta, dado que sólo mide 20,5×16 cm. En el centro hay un cuadrado de fibra de carbón de 11×11 cm que cubre una membrana sensible al tacto similar al teclado del Spectrum. Presionando simplemente un dedo o un lápiz sobre la membrana, el usuario puede guiar un cursor a través de la pantalla. Esta característica contrasta con otras tablillas para gráficos que exigen un "punzón" especial para completar el circuito.

La membrana está compuesta por dos láminas de cables conductores, una sobre el eje horizontal y otra sobre el eje vertical. Cuando se presiona contra la membrana, el aparato detecta qué cables están en contacto y le envía las coordenadas resultantes al ordenador. Encima de la membrana al tacto hay dos botones, uno de los cuales se debe pulsar cuando el usuario desea colorear un punto de la pantalla o seleccionar una de las diversas opciones para pintar que hay disponibles. Se puede utilizar cualquiera de los dos botones.

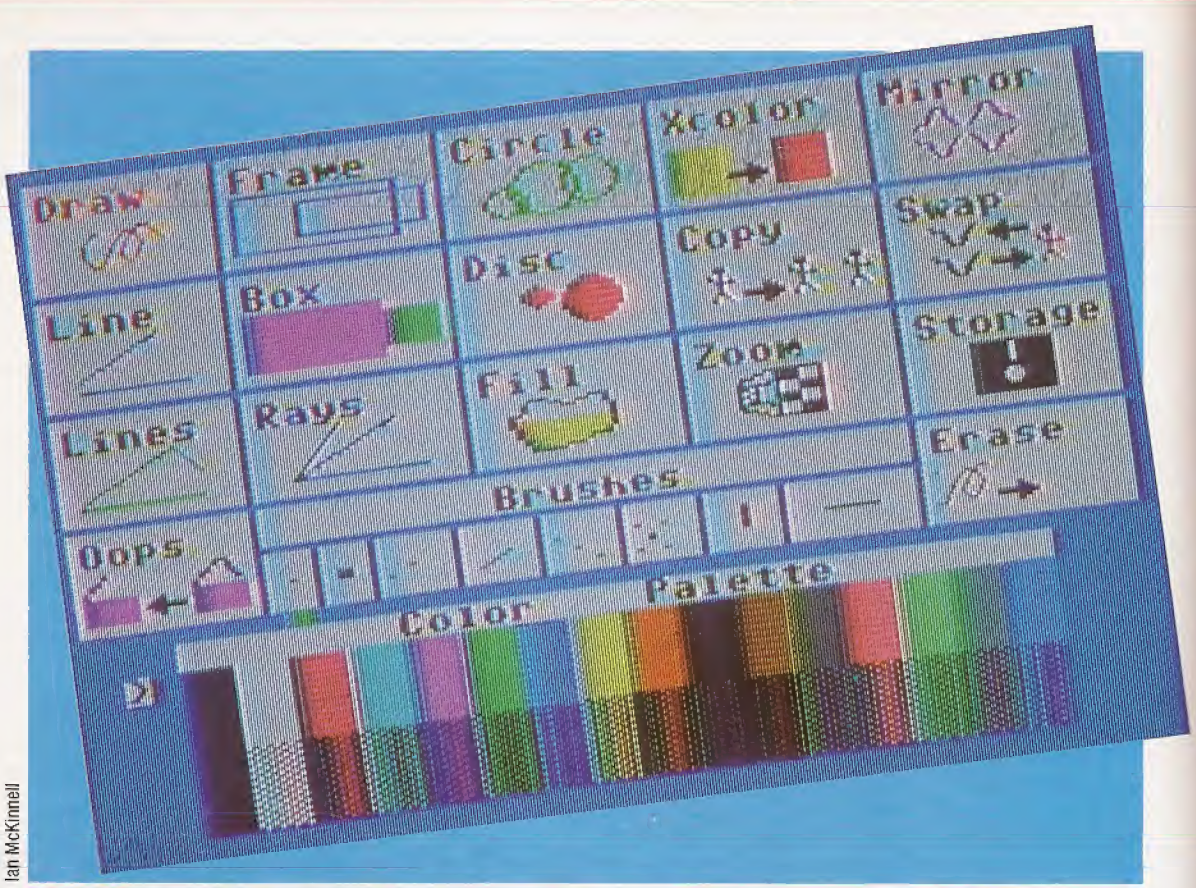
La Koala-pad se conecta con el ordenador a través de la puerta para palanca de mando, y el Koala Painter (el software necesario para operar con el dispositivo) se carga desde disco. Una vez cargado, aparece en la pantalla una visualización de las diversas opciones disponibles. En la parte inferior aparece la "paleta", que contiene 16 colores "puros" y 16 colores de "media tinta". Las medias tintas se obtienen coloreando pixels alternos con distintos colores, lo que proporciona el efecto de matizado. Encima de la paleta hay ocho cajas que contienen los "pinceles". Éstos simplemente se componen de diversas formas que se pueden trazar sobre la pantalla, y van desde un único pixel a combinaciones de pixels y líneas. Rodeando los pinceles se hallan las diversas opciones para dibujar líneas o formas en la pantalla. Éstas las selecciona el usuario presionando contra la membrana y dirigiendo de ese modo una flecha de cursor. Cuando la flecha señala la opción deseada, la pulsación de uno de los botones Select la hace entrar en acción. La opción se vuelva intermitente para recordar al usuario cuál es la modalidad que se está utilizando. La Koala-pad proporciona las facilidades para producir líneas individuales, rayos (líneas dibujadas desde un único punto), recuadros y círculos. Se pueden agregar bloques de color mediante la utilización de la opción "caja" (cuadrados coloreados) o "disco" (círculos coloreados). Se obtiene otro tipo de coloreado mediante el empleo de la instrucción FILL para rellenar un área cerrada con el color elegido. Los colores se alteran empleando la instrucción X-COLOUR.

Se pueden dibujar simultáneamente figuras idénticas mediante la utilización del comando MIRROR. Éste divide la pantalla en cuatro secciones, con el cursor restringido al cuarto superior izquierdo. Todo lo que se trace dentro de ese cuadrante se

Para tener en la mano
Empleando el fácilmente comprensible software conducido por menú, en unas pocas horas se pueden construir pantallas de gráficos complejas y sofisticadas. A diferencia de muchas tablillas para gráficos similares, la Koala-pad se puede sostener en la mano mientras se la utiliza

Ian McKinnell

Haga su elección
El menú principal de la Koala-pad se compone de recuadros que contienen tanto el nombre de la función como un icono explicativo. A pesar de que con los iconos se pretende facilitar la comprensión, tal vez algunas de las ilustraciones sean confusas. Se desplaza la flecha del cursor hasta un recuadro y se pulsa el botón "Select". El nombre del recuadro seleccionado se pondrá intermitente para recordar al usuario cuál es la modalidad que está utilizando



Ian McKinnell

copiará automáticamente en el área correspondiente de los otros tres cuadrantes.

Mediante el empleo de la instrucción ZOOM se consigue un trazado de pixels sumamente detallado. El usuario puede elegir cualquier parte de la pantalla, y la misma se visualiza entonces ampliada en una "ventana" en la parte inferior de la pantalla. Los pixels individuales se visualizan como cuadrados del tamaño de un carácter, de ocho por ocho pixels. Gracias a esta característica la producción de tipos y figuras del tamaño de sprites es rápida y sencilla. Estas figuras se colocan luego en cualquier lugar de la pantalla mediante la instrucción COPY; ésta permite que el usuario defina una zona de la pantalla, cuyo contenido se copia en cualquier otra posición.

Después de haber seleccionado estas opciones, la flecha del cursor se desplaza fuera de la pantalla y se pulsa el botón Select. La pantalla presenta entonces el "lienzo" sobre el cual se creará la imagen requerida. Moviendo la flecha del cursor y pulsando el botón Select, se puede dibujar líneas y formas en cualquier lugar de la pantalla normal para gráficos del Commodore 64.

La calidad de los gráficos producidos por este dispositivo es excelente, rivalizando con pantallas de alta resolución producidas con software comercial. Sin embargo, el nivel de calidad de la instrucción DRAW, para dibujar a mano alzada, es decepcionante (calificativo que también comparten otros paquetes para gráficos). La resolución de la matriz de la membrana no se corresponde con la pantalla de alta resolución del 64, de modo que el punzón (o el dedo) del usuario a menudo no se dirigirá directamente sobre una intersección de la cuadrícula y estará, de hecho, activando dos puntos al mismo tiempo. El ordenador, el tratar de interpretar esto, trazará un punto que lamentablemente no siempre estará en la posición que se pretendía. Esto puede llevar a que lo que planificó como una línea recta aparezca como un garabato confuso. Otra crítica es que, aparte de la instrucción ZOOM, no hay ninguna opción para cambiar el color sin tener que retornar al menú principal. Pero éstas son desventajas poco relevantes que la mayoría de las veces pasan a un plano secundario debido a la velocidad con que se ejecutan las instrucciones LINE y FILL.

Otra limitación del software que se podría haber resuelto mejor es el método empleado para borrar los errores. Cuando se produce una equivocación, el error se elimina de la pantalla utilizando la instrucción OOPS, a la cual se accede desde el menú principal. Sin embargo, el empleo de OOPS borrará todo el trabajo que se haya realizado desde que el usuario saliera por última vez del menú principal. Ello significa que tal vez se borre un trabajo de media hora, simplemente debido a un solo error. La alternativa consiste en eliminar un error utilizando la instrucción ZOOM y corrigiendo el error pixel por pixel, lo que en el caso de una caja o un disco mal colocados lleva cierto tiempo. Una corrección que sería muy bien recibida sería la de restringir el alcance de la instrucción OOPS a la última pulsación del botón Select en vez de a la última salida del menú principal.

Como uno esperaría de un dispositivo que se conecta en la puerta para palanca de mando, se puede utilizar la Koala-pad como una palanca de estas características, permitiendo a los usuarios acceder a la Koala-pad desde sus propios programas. La posición del cursor se puede obtener desde BASIC leyendo (PEEK) las posiciones 54297 y 54298 para las coordenadas X e Y respectivamente.

Las pantallas se pueden guardar en disco y trans-

ferir fácilmente a los propios programas del usuario, permitiendo desarrollar aventuras estilo *Hobbit* con texto en la parte inferior de la pantalla y una imagen arriba. Mediante la utilización de la Koala-pad es posible guardar y recuperar hasta 16 pantallas distintas de ocho Kbytes en un disco. Si bien no es posible cargar una pantalla desde disco directamente en la memoria de pantalla, en la guía para el usuario se ha incluido un programa que permite la transferencia de pantallas desde la zona en la cual están cargadas a la memoria de pantalla.

La resolución máxima de una pantalla Koala Painter guardada es de 255×255 pixels si está incluida en un programa en BASIC, si bien utilizando código máquina se pueden conseguir resoluciones mayores. Esta limitación podría plantear problemas, dado que la pantalla en alta resolución del Commodore 64 en realidad tiene unas dimensiones de 320×200 pixels. La Koala-pad está limitada a un máximo de 255 pixels porque éste es el número mayor que puede direccionar un byte; para direccionar una pantalla completa se requerirían direcciones de 16 bits. Lamentablemente, esto puede llevar a que el usuario pierda una porción de la visualización. Esto es así porque cuando la pantalla no está siendo visualizada, estará almacenada en algún lugar dentro de la RAM del usuario. Por lo tanto, la pantalla de ocho Kbytes, junto con la información de color, se debe transferir desde su posición en la memoria para el usuario hasta la memoria de pantalla en alta resolución, que empieza en la posición 55296.

KOALA-PAD

DIMENSIONES
210×165 mm

PANTALLA
Se utiliza la visualización en pantalla completa del Commodore 64, de 320×200, si bien ésta se puede ver restringida a 255×255 al llamar una pantalla desde un programa en BASIC

INTERFACE
Se conecta a través de la puerta para palanca de mando del Commodore 64

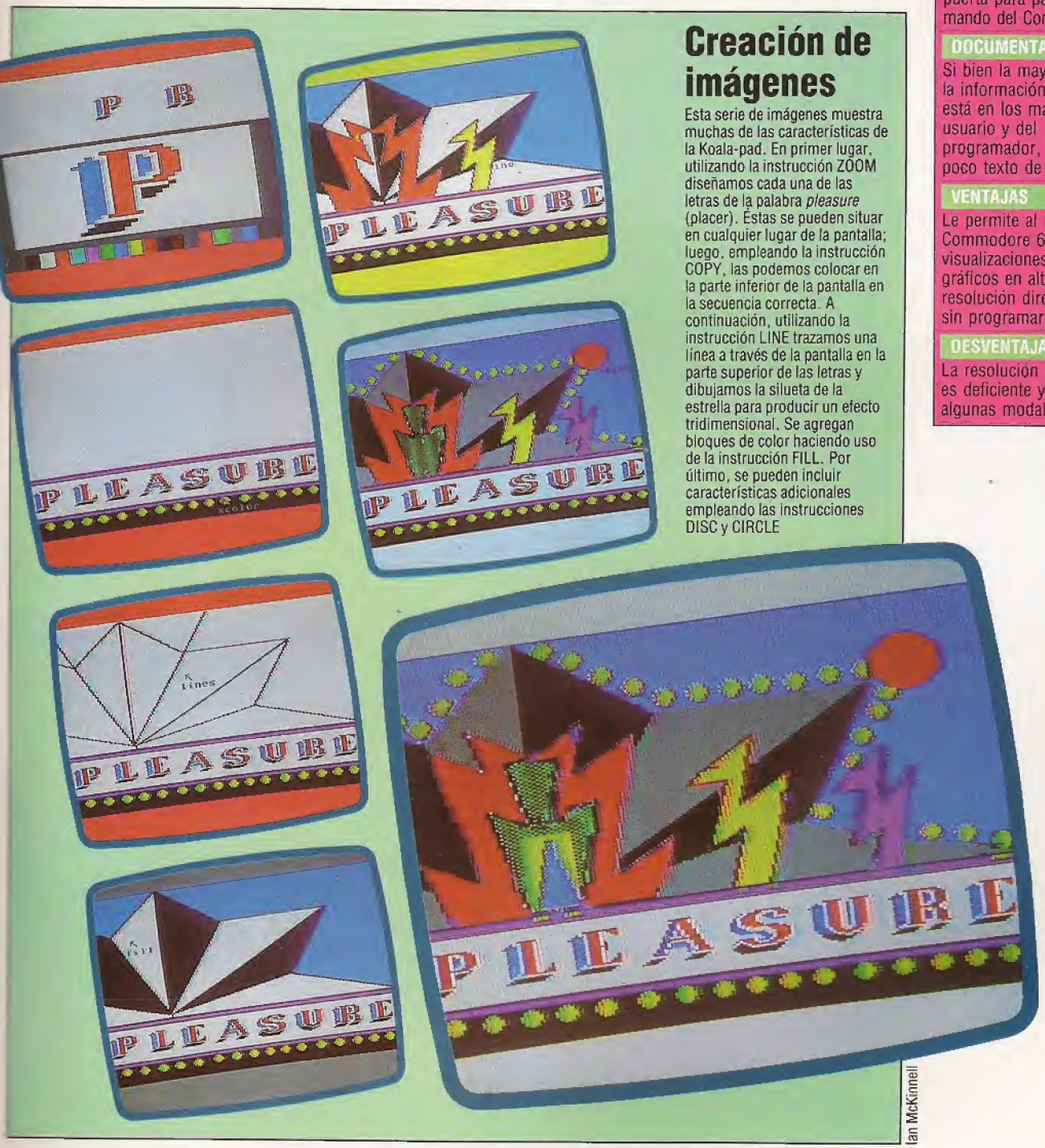
DOCUMENTACION
Si bien la mayor parte de la información requerida está en los manuales del usuario y del programador, hay muy poco texto de explicación

VENTAJAS
Le permite al usuario del Commodore 64 producir visualizaciones de gráficos en alta resolución directamente, sin programar

DESVENTAJAS
La resolución de la tablilla es deficiente y difusa en algunas modalidades

Creación de imágenes

Esta serie de imágenes muestra muchas de las características de la Koala-pad. En primer lugar, utilizando la instrucción ZOOM diseñamos cada una de las letras de la palabra *pleasure* (placer). Éstas se pueden situar en cualquier lugar de la pantalla; luego, empleando la instrucción COPY, las podemos colocar en la parte inferior de la pantalla en la secuencia correcta. A continuación, utilizando la instrucción LINE trazamos una línea a través de la pantalla en la parte superior de las letras y dibujamos la silueta de la estrella para producir un efecto tridimensional. Se agregan bloques de color haciendo uso de la instrucción FILL. Por último, se pueden incluir características adicionales empleando las instrucciones DISC y CIRCLE



Ian McKinnell

En dos ruedas

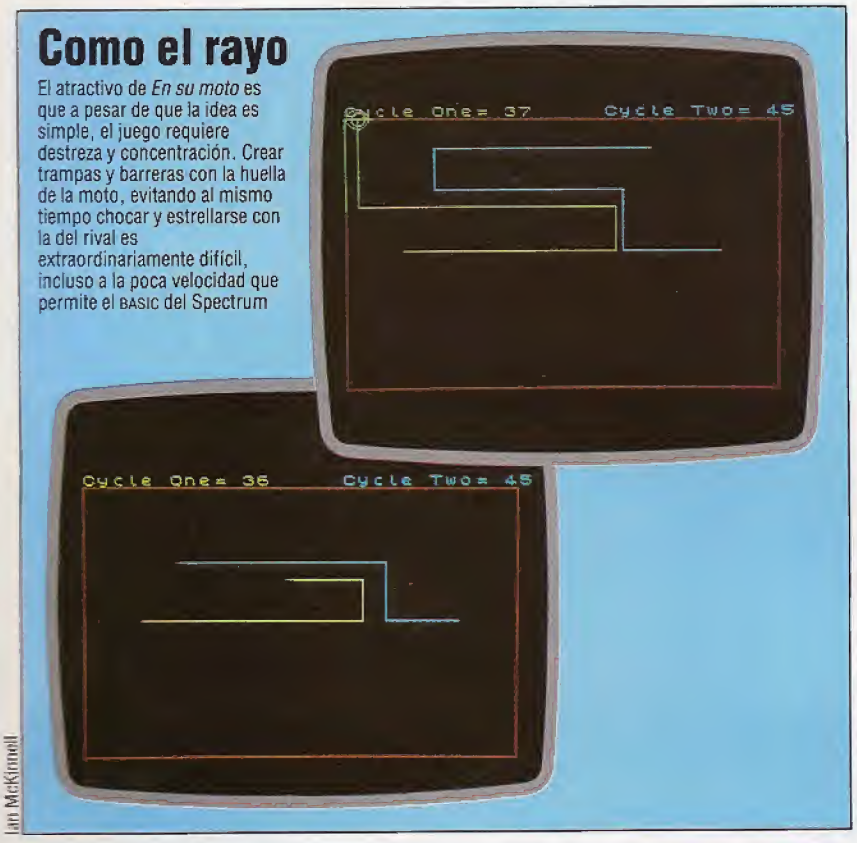
El juego que presentamos es muy simple y consta de sólo 35 líneas de BASIC, pero aun así resulta muy atractivo

El juego se llama *En su moto* y está basado en una escena de la película *Tron*, producida por la firma Walt Disney. Es una contienda entre dos contrincantes que exige destreza y reacciones rápidas y tiene lugar en un circuito cerrado. Cada jugador tiene una motocicleta que viaja a una velocidad desbocada y no se la puede detener. Su único control permite girar 90° a la velocidad máxima. Estas motocicletas dejan tras de sí paredes sólidas de luz, y el objetivo del juego consiste en forzar al contrincante a estrellarse con el laberinto que uno va creando mientras se desplaza a toda velocidad.

El juego se ha implementado en el ZX Spectrum, que no se destaca por la velocidad de su BASIC. Dado que se trata de un juego de acción, el programa se ha diseñado atendiendo más a la velocidad que a la elegancia, por lo que gran parte del listado puede parecer algo desestructurado.

Como el rayo

El atractivo de *En su moto* es que a pesar de que la idea es simple, el juego requiere destreza y concentración. Crear trampas y barreras con la huella de la moto, evitando al mismo tiempo chocar y estrellarse con la del rival es extraordinariamente difícil, incluso a la poca velocidad que permite el BASIC del Spectrum



Ian McKinnell

Se han evitado llamadas a subrutinas y otros dispositivos estructurados, porque habrían supuesto sacrificar velocidad de ejecución.

La primera etapa es el diseño del circuito y de la visualización del marcador. Como puede verse, éste es bastante sencillo, lo que contribuye a la brevedad del programa final. El único punto que hay que destacar es que el borde del circuito corresponde a un carácter dentro de la zona de pantalla utilizable. Esto es para asegurar que los gráficos resultantes de una colisión con la pared del circuito no se salgan de pantalla:

```
 10 LET p=0:LET q=0
100 BORDER 0:PAPER 0:CLS
110 PRINT AT 0,1;INK 6;"Moto uno=";q
120 PRINT AT 0,19;INK 5;"Moto dos=";p
130 INK 2
140 PLOT 8,8:DRAW 239,0:DRAW 0,159
150 DRAW -239,0:DRAW 0,-159
```

El circuito se ha dibujado en rojo, y hemos elegido el amarillo para representar a la motocicleta uno y el cyan (azul claro) para la motocicleta dos. Las variables *p* y *q* retienen el marcador en curso de los dos rivales.

La siguiente etapa es la inicialización de las variables, y aquí debemos empezar a pensar acerca de cómo implementaremos la acción principal del juego. La acción para una única motocicleta es bastante directa y está reflejada en el diagrama de flujo (p. 1113). Utilizando POINT verificamos si la posición en curso de la moto está ocupada y, de ser así, pasamos a la rutina de colisión. Si no lo está, pasamos a esa posición mediante PLOT y después leemos el teclado para comprobar cualquier cambio de dirección. Nuestra posición se incrementa, entonces, en uno en la dirección actual, y el ciclo vuelve a comenzar. Por tanto, necesitamos dos variables: dos para las coordenadas *x* e *y* en curso, y dos para la dirección en curso a lo largo de los ejes *x* e *y*.

No obstante, se trata de dos motos que se mueven al mismo tiempo. Una solución elegante sería utilizar cuatro matrices de dos elementos *x(2)* e *y(2)*, para las posiciones, por ejemplo, pero esto retardaría el juego, de modo que tenemos que emplear ocho variables separadas:

```
200 LET x=40:LET y=88
210 LET m=215:LET n=88
220 LET a=1:LET b=0
230 LET i=-1:LET j=0
```

Esto establece las posiciones iniciales de las motos y hace que se muevan a un pixel por vez. La acción básica es, por tanto, bastante fácil de implementar:

```
400 IF PONT(x,y)=1 THEN LET col=6:GOTO 700
410 IF POINT (m,n)=1 THEN LET col=5:LET x=m:
    LET y=n:GOTO 700
420 PLOT INK 6;x,y:PLOT INK 5;m,n
```

(entre las líneas 500 y 570 está la rutina de teclado que establece nuevos valores para *a*, *b*, *i* y *j*):.

```
600 LET x=x+a:LET y=y+b
610 LET m=m+i:LET n=n+j
620 GOTO 400
```

Quizá el único punto que se preste a confusión sea la línea 410, donde se establecen para la moto número dos las variables de la moto número uno, y se introduce una nueva variable, col. Ello es así para que se pueda utilizar una única rutina para la acción de la colisión, donde *x* e *y* se emplean simplemente para indicar el punto en el cual tiene lugar la colisión, y col establece el color.

La rutina para verificar el teclado ha de ser rápida, pero lamentablemente no hemos tenido más remedio que utilizar sentencias IF...THEN que son más bien lentas. No obstante, podemos utilizar la veloz instrucción IN para leer las teclas. Las teclas de control elegido son Q y A, que controlan el movimiento hacia arriba y hacia abajo de la moto uno, y P y ENTER para la moto dos. Izquierda y derecha son X y C para la moto uno, y N y M para la moto dos. (Véase p. 846 para una explicación completa acerca de cómo se relacionan los bloques de teclas con los bytes de la memoria.)

```
500 IF IN 64510=190 THEN LET a=0:LET b=1
510 IF IN 65022=190 THEN LET a=0:LET b=-1
520 IF IN 65278=187 THEN LET a=-1:LET b=0
530 IF IN 65278=183 THEN LET a=1:LET b=0
540 IF IN 57342=190 THEN LET i=0:LET j=1
550 IF IN 49150=190 THEN LET i=0:LET j=-1
560 IF IN 32766=187 THEN LET i=1:LET j=0
570 IF IN 32766=183 THEN LET i=-1:LET j=0
```

Todo lo que resta es la rutina de la colisión y la actualización de los marcadores. Se eligió una serie de círculos concéntricos que se van ampliando, centrados en el punto del impacto, con radios de cuatro, seis y ocho pixels:

```
700 FOR d=1 TO 3
710 CIRCLE BRIGHT 1;INK col;x,y,2+d*2
720 NEXT d
730 IF col=6 THEN LET p=p+1:GOTO 750
740 LET q=q+1
750 GOTO 100
```

Con esto se acaba el juego, con la última sentencia saltando hacia atrás hasta los procedimientos de inicialización del principio. Al juego, sin embargo, le iría bien un procedimiento de inicio para facilitar más su utilización:

```
300 PRINT AT 10,5;INK 7;"PULSE CUALQUIER
    TECLA PARA EMPEZAR"
310 IF INKEY$="" THEN GOTO 310
320 PRINT AT 10,5;"                    "
```

Esto proporciona una pausa entre rondas consecutivas. Sólo resta guardar el juego en cassette, utilizando de preferencia SAVE"En su moto"LINE10, de modo que el juego se ejecute automáticamente apenas se cargue.

Evidentemente se podría hacer que el juego fuera más emocionante, con pantallas de instrucciones, pantallas de carga, una opción para un solo jugador con una rutina de estrategia que controle la otra motocicleta del modo que hemos sugerido, sonido y mejores gráficos. Pero estas últimas opciones harán que el juego se desarrolle de manera excesivamente lenta.

En un próximo capítulo volveremos a escribir *En su moto* en lenguaje máquina con el fin de demostrar su auténtico potencial.

Hombre contra máquina

Ajustes

En la versión para un solo jugador el ordenador es el jugador 1. Se omite la sección del programa que verifica las teclas de instrucciones del Jugador 1, y el algoritmo del código proporcionado permite que el Jugador 1 continúe moviéndose en una línea recta hasta que se genere un movimiento ilegal o hasta que se mueva el Jugador 2. En este último caso, este movimiento es reflejado a modo de espejo o copiado, comprobándose luego su legalidad. Cada vez que se genera un movimiento ilegal del Jugador 1, el programa busca movimientos legales en ángulos rectos respecto a la dirección del movimiento ilegal; de no hallarse ninguno, la posición es desesperada.

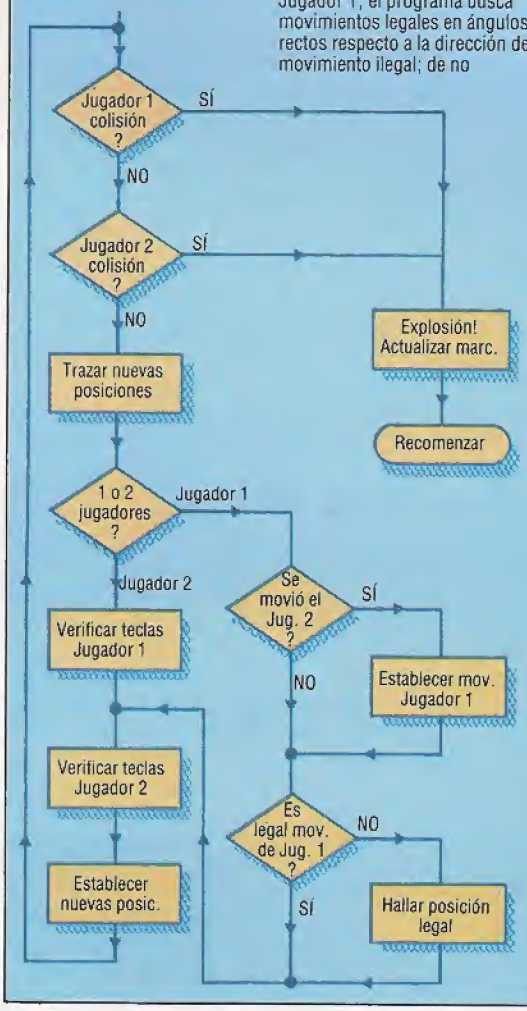
Existen muchas maneras de implementar una versión de este juego para un único jugador. Los cambios que sugerimos hacen que sea posible elegir entre las versiones para un solo jugador y para dos jugadores al comienzo de cada juego. No es difícil inventar algoritmos para ejecutar este juego, pero sí lo es implementarlos en BASIC sin retardar el juego. Introduzca estas modificaciones:

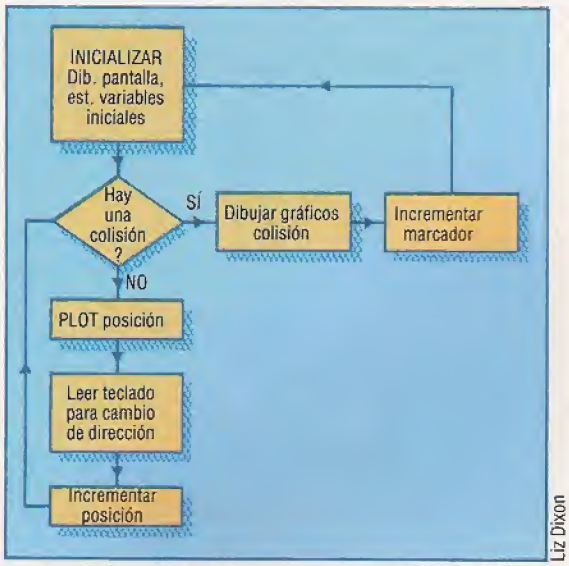
```
20 LET teclado=500:LET
   pt=-1:LET inmovil=0:
   RANDOMIZE
260 PRINT AT 10,5;"N. de
   jugadores (1/2)?"
270 LET A$=INKEY$:IF
   a$<>"1" AND a$<>"2"
   THEN GO TO 270
280 IF INKEY$<>"2" THEN
   LET teclado=440
```

Estas líneas le ofrecen al usuario la selección del tipo de juego e implementan la elección accediendo o bien a la estrategia para un solo jugador entre las líneas 440 y 460, o a la versión estándar para dos jugadores entre las líneas 500 y 570. He aquí nuestra estrategia:

```
430 GOTO teclado
440 LET pt=SGN(RND-0.5):IF
   inmovil=pt THEN LET
   a=pt*j:
   LET b=pt*i:LET inmovil=0
450 IF POINT (x+a,y+b)<>1
   THEN GOTO 540
460 LET pt=SGN(RND-.5):LET
   a=a*pt:LET b=b*pt:LET
   d=a:LET a=b:LET b=d:IF
   POINT (x+a,y+b)<>1
   THEN GOTO 540
490 LET a=-a:LET b=-b:
   GOTO 540
540 IF IN 57342=190 THEN LET
   i=0:LET j=1:LET
   inmovil=1
```

Y agregue, igualmente:
:LET inmovil=1
al final de las líneas 550 a 570





Liz Dixon

Para un solo jugador
Este diagrama de flujo simplificado ilustra la estructura del programa con un solo jugador. Cada proceso se repite en el juego en versión para dos jugadores

Bajo su control

He aquí cómo controlar un coche Lego con una palanca de mando, utilizando los dos dispositivos creados en capítulos anteriores

Las palancas de mando se utilizan de manera muy diferente en el Commodore 64 y el BBC Micro. El primero emplea una palanca de mando estándar tipo Atari que opera mediante cuatro interruptores direccionales y un pulsador de disparo. Contrastando con esta disposición digital, el BBC Micro utiliza una palanca de mando analógica, o paleta. Esta clase de palanca no se basa en el simple hecho de hacer o no hacer contactos, sino que emplea dos potenciómetros, uno para el movimiento izquierda-derecha y el otro para el movimiento arriba-abajo (si bien una palanca de tipo digital se puede adaptar para su uso con la puerta analógica del BBC). Dado que estos métodos de funcionamiento son tan distintos, los abordaremos por separado.

Commodore 64

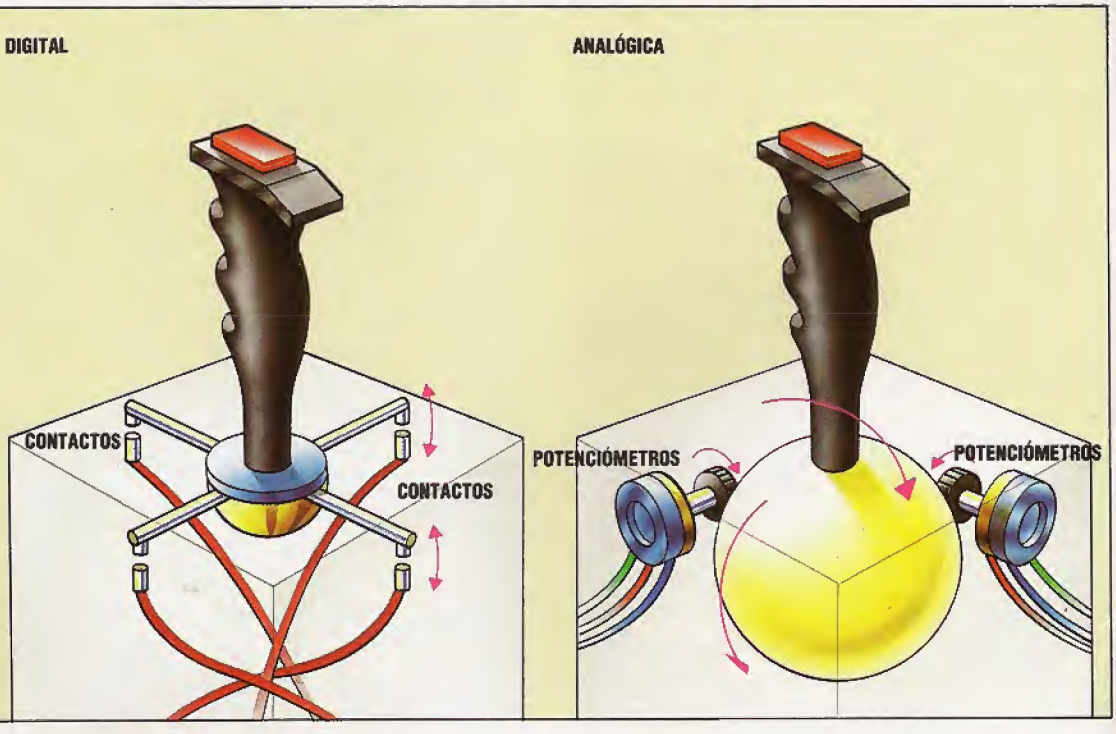
La palanca de mando tipo Atari que utiliza el Commodore 64 se enchufa en una de las puertas para juegos, situada junto al interruptor de potencia on/off. En la explicación y el programa siguientes emplearemos la puerta 2, de modo que si dispone de una palanca de mando enchúfela en la puerta 2 (la que está más cerca del interruptor on/off). Si ésta se desplaza desde la posición central hacia arriba, se cerrará uno de los cuatro interruptores internos. La puerta 2 se une directamente con la posición de memoria 56320, y el cierre de este interruptor hará que uno de los bits de esta posición se ponga a 0, más o menos de la misma forma en que el cerrar un interruptor externo conectado a la caja buffer haría que un bit del registro de datos de la puerta para el usuario se pusiera a 0. Entre el breve programa que reseñamos a continuación, que visualiza repetidamente el valor del registro de datos. Con el programa en marcha desplace la palanca de mando hacia uno y otro lado y accione el pulsador de disparo, observando las diversas modificaciones que se producen en el valor visualizado en la pantalla.

```
 10 REM **** LEER PALANCA MANDO CBM 64 ****
 20 PUERTA2=56320
 30 PALANCA=PEEK(PUERTA2):GOSUB500
 40 PRINT CHR$(145);PALANCA,B$
 50 GOTO30
 60 :
500 REM S/R CONVERSION A BINARIO
510 B$="":N=PALANCA
520 FOR D=1 TO 8
530 N1=INT(N/2):R=N-2*N1
540 B$=B$+STR$(R):N=N1
550 NEXT D
560 RETURN
```

Al cabo de algunos minutos de experimentación ya debería resultar claro qué bits de la posición de la palanca de mando corresponden a los cuatro interruptores de dirección y al pulsador de disparo. Normalmente, con la palanca en la posición central, el contenido de la posición de ésta es 127, es decir, 01111111. Al empujar la palanca, el valor pasa a 126 (01111110). El bit 0 está relacionado con el interruptor de dirección hacia arriba. Los efectos

Procedimientos diferentes

La palanca de mando digital, o "de contacto", simplemente registra el movimiento del bastón en dos direcciones cualesquiera de las cuatro principales; la cantidad de movimiento no se mide. La palanca de mandos analógica emplea potenciómetros perpendicularmente opuestos para medir el movimiento del bastón, y comunica la información como dos voltajes variables. Éstos se convierten luego en un número mediante el convertidor de analógico a digital



Kevin Jones

de los interruptores de la palanca en la posición de memoria de ésta son los siguientes:

Palanca de mando	Decimal	Binario
Central	127	01111111
Arriba	126	01111110
Abajo	125	01111101
Izquierda	123	01111011
Derecha	119	01110111
Disparo	111	01101111

Quizá también haya observado que el desplazamiento de la palanca en diagonal hace que dos interruptores se cierren simultáneamente. A pesar de que para controlar nuestro vehículo no necesitamos la detección del movimiento en diagonal, los resultados de tales desplazamientos son éstos:

Palanca de mando	Decimal	Binario
Arriba e izquierda	122	01111010
Arriba y derecha	118	01110110
Abajo e izquierda	121	01111001
Abajo y derecha	117	01110101

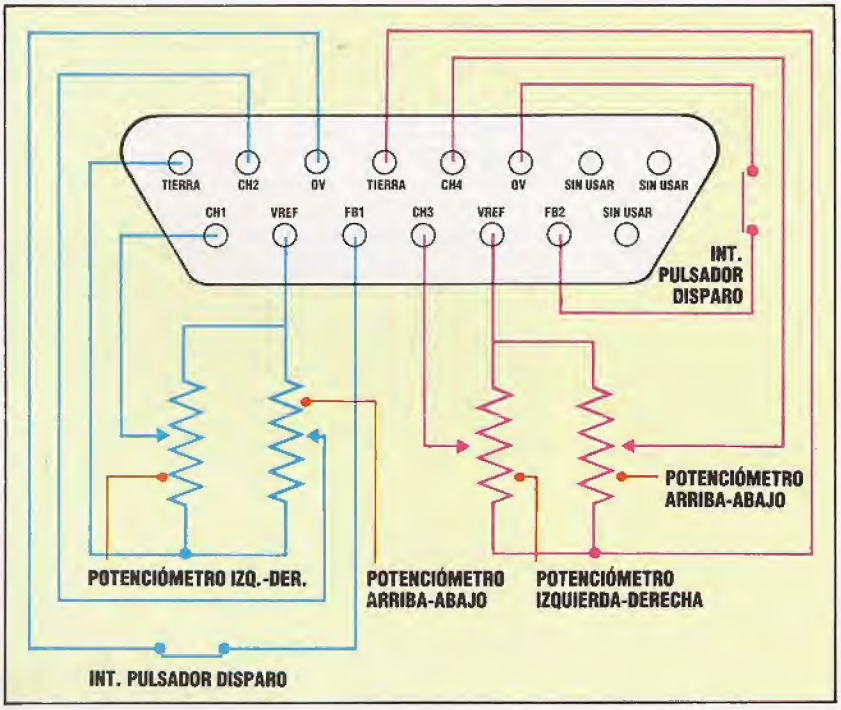
El siguiente programa utiliza una palanca de mando para controlar los movimientos del vehículo de motores gemelos. El vehículo se deberá conectar a la caja de salida de la misma forma que en la página 1066, y la palanca se deberá enchufar en la puerta para juegos 2, situada en el lado derecho del ordenador.

```
  10 REM PALANCA DE MANDO CBM 64
  20 RDD=56579:REGDAT=56577
  30 POKERDD,255:REM TODAS SALIDA
  40 PALANCA=PEEK(56320):REM PALANCA PUERTA 2
  50 GOSUB1000:REM COMPROBAR PALANCA DE MANDO
  60 POKEREGDAT,0:GOTO40
  90 :
1000 REM S/R COMPROBACION PALANCA MANDO
1005 IF PALANCA=127 THEN POKEREGDAT,0
1010 IF PALANCA=126 THEN POKEREGDAT,5
1020 IF PALANCA=125 THEN POKEREGDAT,10
1030 IF PALANCA=123 THEN POKEREGDAT,6
1040 IF PALANCA=119 THEN POKEREGDAT,9
1050 IF PALANCA=111 THEN POKEREGDAT,0:END
1060 RETURN
```

BBC Micro

La palanca de mando del BBC es un dispositivo analógico que se basa en dos potenciómetros para proporcionar información sobre los movimientos efectuados hacia arriba-abajo e izquierda-derecha. La diferencia fundamental entre una palanca de mando digital y una de tipo analógico como la que emplea el BBC Micro reside en que la última proporciona información acerca de la "posición" entre límites dados, mientras que la primera sólo proporciona información acerca de la dirección del recorrido. Los potenciómetros trabajan así:

Un potenciómetro es básicamente una resistencia a través de la cual se aplica un voltaje. Un tercer conector del potenciómetro se puede desplazar a lo largo de la resistencia tomando una fracción del voltaje suministrado. Esta fracción depende de la posición de la conexión móvil. En un tipo lineal de potenciómetro, si la conexión móvil se colocara a medio camino a través de la resistencia, los voltajes derivados serían la mitad del voltaje suministrado. Por consiguiente, mediante el desplazamiento de la conexión central se puede obtener cualquier voltaje entre cero y el voltaje suministrado. Girar el mando del volumen de una radio es, esencialmente, mover la conexión del medio a través de la resistencia del potenciómetro de control de volumen. En una palanca analógica el movimiento de la conexión del medio se realiza moviendo la paleta.



Reflexiones analógicas
Las palancas de mando para el BBC Micro por lo general se suministran por parejas, que van a un único conector. Las salidas de las patillas de la puerta analógica, tal como se ven desde fuera de la máquina, muestran que la palanca 2 está conectada de modo similar a la palanca 1. En el lugar de cualquiera de las dos se pueden conectar palancas individuales

Las palancas de mando para el BBC por lo general se venden a pares. En el diagrama superior vemos las conexiones de la puerta analógica del BBC Micro para las palancas 1 y 2.

El micro proporciona un voltaje de referencia a través de cada potenciómetro y el voltaje derivado de la conexión del medio se alimenta a través de las entradas de dos canales. El canal 1 se utiliza para la entrada desde el potenciómetro izquierda-derecha, y el canal 2 para aceptar la entrada desde el potenciómetro arriba-abajo. El pulsador de disparo es un simple interruptor para abrir o cerrar contacto.

Una vez se han aceptado las entradas del potenciómetro, se las debe convertir de la forma analógica a la digital mediante un convertidor interno. Esta conversión se realiza comparando el voltaje de entrada con el voltaje de referencia, y el tiempo de conversión es de alrededor de 10 milisegundos para la lectura de cada canal. Después de que la señal de entrada de la palanca está en forma digital, podemos utilizar los valores para controlar el vehículo.

La entrada a la puerta analógica se puede leer desde BASIC mediante el empleo de la instrucción ADVAL. El valor que devuelve ADVAL está comprendido entre 0 y 65520, correspondiendo el límite superior a una entrada igual al voltaje de referencia. Las reducciones del voltaje de entrada irán produciendo números correspondientemente más pequeños hasta que un voltaje de entrada de cero voltios produzca la devolución por parte de ADVAL de un valor de cero. Para nuestra aplicación sólo interesan los dos valores límite. El canal leído por ADVAL está determinado por el número entre paréntesis que sigue a la palabra clave. Por tanto, ADVAL(1) leerá el canal 1 y devolverá un valor que estará comprendido entre 0 y 65520.

ADVAL(0) cumple dos funciones diferentes. Los dos bits menos significativos corresponden a los pulsadores de disparo de la palanca de mando 1 y la palanca de mando 2. X=ADVAL(0) AND 3 devolverá un valor de uno si se acciona el pulsador de disparo de la palanca 1. X=ADVAL(0) DIV 256 dará el número del canal que ha completado en último lugar una conversión de analógico a digital (A a D).

Dado que la conversión de cada canal de entrada analógica lleva alrededor de 10 milisegundos, el proceso de cada uno de los cuatro canales llevará 40 milisegundos. En nuestra aplicación sólo empleamos los canales 1 y 2. Podemos reducir el tiempo de conversión utilizado especificando qué canales requieren conversión. Esto se puede hacer por medio de *FX16,2, que habilita los canales 1 y 2 pero inhabilita los canales 3 y 4. El siguiente programa combina toda esta información.

```
10 REM CONTROL DE PALANCA BBC
20 RDD=&FE62:REGDAT=&FE60
30 ?RDD=255:REM TODAS SALIDA
40 REM HABILITAR CANALES A-D 1 y 2
50 *FX16,2
60 REPEAT
70 PROCleer—palanca
80 UNTIL disparo=1
90 END
100 :
110 DEF PROCleer—palanca
120 REPEAT
130 canal=ADVAL(0) DIV 256
140 UNTIL canal<>0:REM ESPERAR CONVERSION
150 IF canal=1 THEN PROCizquierda—derecha
160 IF canal=2 THEN PROCarriba—abajo
170 ENDPROC
180 :
190 DEF PROCizquierda—derecha
200 REPEAT
210 valpal=ADVAL(1)
220 IF valpal<100 THEN ?REGDAT=9
230 IF valpal>64000 THEN ?REGDAT=6
240 disparo=ADVAL(0) AND 3
250 PRINT?REGDAT,canal,valpal
260 UNTIL(valpal>100 AND valpal<64000) OR disparo=1
270 ?REGDAT=0
280 ENDPROC
290 :
300 DEF PROCarriba—abajo
310 REPEAT
320 valpal=ADVAL(2)
330 IF valpal<100 THEN ?REGDAT=10
340 IF valpal>64000 THEN ?REGDAT=5
350 disparo=ADVAL(0) AND 3
360 PRINT?REGDAT,canal,valpal
370 UNTIL(valpal>100 AND valpal<64000) OR disparo=1
380 ?REGDAT=0
390 ENDPROC
```

Respuestas a los ejercicios

1) La calibración de su vehículo se puede efectuar cronometrando el tiempo invertido para recorrer varias distancias, comúnmente 10 cm, 20 cm, 50 cm, 100 cm y 150 cm. Calculando la velocidad sobre cada distancia y haciendo un promedio, se puede obtener una buena estimación para la distancia recorrida en un segundo. Este valor se puede utilizar para controlar el vehículo sobre distancias medidas. Se podría adoptar un enfoque similar para girar, seleccionando una cantidad de ángulos y efectuando cronometrajes. Controlar motores encendiéndolos y apagándolos durante períodos de tiempo medidos puede plantear muchas dificultades, sin contar con que la estructura del programa controlador es tal que los intervalos se deben medir con la mayor exactitud. Diferencias de centésimas de segundo pueden producir enormes divergencias en las distancias o los ángulos de giro.

2) El listado del programa ofrecido en la página 1093 permitirá conducir el vehículo a través del recorrido con obstáculos. Seguir el patrón en sentido contrario es un poco más complicado. Primero debemos asignar un par de variables para cada dirección junto con su inverso. De modo que, por ejemplo, adelante y marcha atrás se agrupan en el mismo par.

Commodore 64

```
17 A(1)=5:B(1)=10:A(2)=10:B(2)=5
18 a(3)=6:B(3)=9:A(4)=9:B(4)=6
```

Entonces se pueden agregar las siguientes rutinas para reproducir la secuencia grabada al revés:

```
92 GOSUB 2000: REM REPRODUCCION INVERTIDA
2000 S/R REPRODUCCION INVERTIDA
2010 FOR I=C TO 1 STEP -1
2020 FOR J=1 TO 4
2030 IF DR(I,1)=A(J) THEN POKE REGDAT,B(J):J=4
2040 NEXT J
2050 T=TI
2060 IF (TI-T)<DR(I,2) THEN 2060
2070 NEXT I
2080 STOP
2090 RETURN
```

BBC Micro

```
1020 DIM DR(100,2),A(10),B(10)
1025 A(1)=5:B(1)=10:A(2)=10:B(2)=5
1026 A(3)=6:B(3)=9:A(4)=9:B(4)=6
1115 PROCreproduccion—invertida
2000 DEF PROCreproduccion—invertida
2010 FOR I=C TO 1 STEP -1
2020 FOR J=1 TO 4
2030   IF DR(I,1)=A(J) THEN?REGDAT=B(J):J=4
2040 NEXT J
2042 TIME=0
2045 REPEAT UNTIL TIME>=DR(I,2)
2047 ?REGDAT=0
2050 NEXT I
2055 PRINT"PULSE C PARA CONTINUAR"
2060 REPEAT A$=GET$
2070 UNTIL A$="C"
2080 ENDPROC
3000 FORI=1TOC:PRINTDR(I,1),DR(I,2)
3010 NEXT
```

3) Suponiendo que la línea 7 es adelante, la línea 6 es atrás, la 5, izquierda y la 4, derecha:

```
10 REM INTERRUPTORES EXTERNOS BBC
20 RDD=&FE62:REGDAT=&FE60
30 ?RDD=15:REM LINEAS 4-7 ENTRADA
40 ?REGDAT=0
50 PROCleer—entrada
60 GOTO 50
70 :
80 DEF PROCleer—entrada
90 IF(?REGDAT AND 240)=240 THEN ?REGDAT=0
100 IF(?REGDAT AND 128)=0THEN ?REGDAT=5
110 IF(?REGDAT AND 64)=0THEN ?REGDAT=10
120 IF(?REGDAT AND 32)=0THEN ?REGDAT=6
130 IF(?REGDAT AND 16)=0THEN ?REGDAT=9
140 ENDPROC
```

```
10 REM INTERRUPTORES EXTERNOS CMB 64
20 RDD=56579:REGDAT=56577
30 POKERDD,15:REM LINEAS 4-7 ENTRADA
40 POKEREGDAT,0:REM MOTORES APAGADOS
50 REM L7 ADEL.,L6 ATR.,L5 IZQ.,L4 DER
55 REM ESTA BAJA UNA DE LAS LINEAS DE ENTRADA?
60 IF(PEEK(REGDAT)AND240)<>240 THENGOSUB1000:GOTO60
70 POKEREGDAT,0:GOTO60
80:
1000 REM S/R EXPLORACION LINEAS ENTRADA
1005 IF(PEEK(REGDAT)AND128)=0THENPOKEREGDAT,5
1010 IF(PEEK(REGDAT)AND64)=0THENPOKEREGDAT,10
1020 IF(PEEK(REGDAT)AND32)=0THENPOKEREGDAT,6
1030 IF(PEEK(REGDAT)AND16)=0THENPOKEREGDAT,9
1040 RETURN
```

Saltos y tablas

Presentamos un análisis del direccionamiento indexado

Ante todo hay que dejar bien sentado que el direccionamiento indirecto no es un modo más de direccionar sino un recurso adicional que se emplea juntamente con casi todos los demás modos; en realidad se trata de un paso ulterior en el cálculo de la dirección efectiva (la dirección de donde efectivamente se toman los datos). La dirección efectiva o real se calcula de alguna de las maneras que hemos descrito, pero si se escoge el direccionamiento indirecto implica que el contenido de la dirección así calculada es considerado a su vez como una dirección, junto con la siguiente posición consecutiva de la memoria. Esta dirección se convierte en dirección efectiva final, y de ella se extraen los datos.

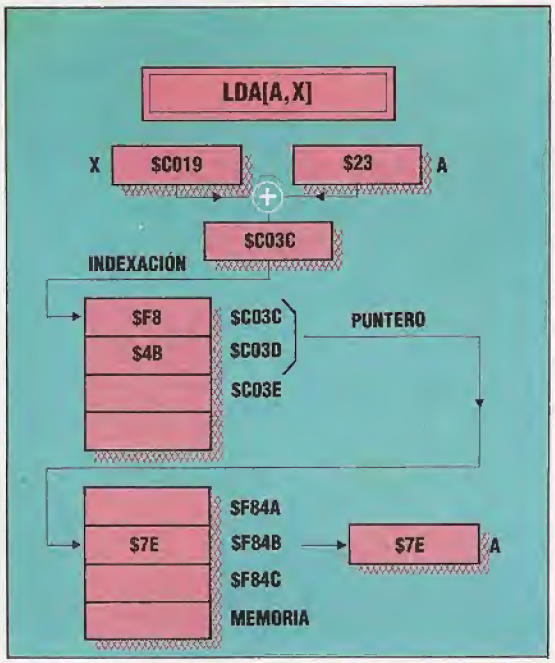
Si los siguientes valores son almacenados:

Dirección	Contenido
3000	40
3001	0A
400A	F2

entonces la instrucción LDA $3000 cargará el valor $40 en el acumulador A, siendo la dirección efectiva $3000. El modo indirecto siempre queda especificado por medio de corchetes que encierran el operando, así LDA [$3000] cargará el valor $F2 en A, y la dirección efectiva es el valor almacenado en la dirección que a su vez quedó almacenado en $3000 y $3001, en nuestro caso, $400A. Es de notar que para el 6809 se conviene en que las direcciones sean almacenadas con el byte *hi* precediendo al byte *lo*: así $40 se almacena en $3000 y $0A en $3001. Es lo que se conoce por el convenio *hi-lo*. Justo el convenio opuesto al empleado por el Zilog Z80 y el MOS Tech 6502: en éstos el byte *lo* de la dirección, $0A, se almacenaría en $3000 y el byte *hi*, $40, en $3001.

El direccionamiento indirecto a menudo se potencia si se usa con el indexado. La instrucción LDA [A,X], que está en el modo indexado indirecto, calculará primero una dirección sumando los contenidos de A y de X y después utilizará el valor de 16 bits almacenado en esta posición y la siguiente como una dirección cuyo contenido se cargará en A.

De hecho, el 6809 usa menos el direccionamiento indirecto que muchos procesadores (los programas tanto para el 6502 como para el Z80 lo utilizan profusamente) dada la potencia de sus modos de direccionamiento indexados. Pero hay ocasiones en que el modo indirecto resulta útil. Una de ellas, que nos disponemos a emplear largamente en un próximo capítulo, es el uso de dispositivos de interface de periféricos. Los procesadores Motorola, al contrario que las familias 8080 y 8086 del Intel, tienen una E/S expresada como un mapa en la memoria. Los registros de comunicaciones introducidos en los dispositivos de interface aparecen dentro del mapa principal de la memoria del sistema, y los valores pueden almacenarse o sacarse de ellos como si fueran unas posiciones de memoria cualesquiera en vez de ser, efectivamente, un conducto hacia el dispositivo de la interface. Una rutina para controlar uno de estos dispositivos (p. ej., una rutina de impresión) necesita la dirección del registro de la interface. Si el dispositivo se reubica en el mapa de memoria, o si no hay más que un dispositivo de ese tipo, entonces es mucho más sencillo tratarlo por medio del cambio de una posición de memoria, la que contiene la dirección del registro de comunicación del dispositivo (un puntero al dispositivo), mejor que recurrir a encontrar y cambiar cada ocurrencia de su dirección. La rutina referencia el dispositivo indirectamente, por medio del puntero.



Direccionamiento indexado indirecto
El argumento [A,X] de la instrucción LDA está entre corchetes, lo que equivale a decir que el contenido de X (aquí,$C019) será añadido al contenido de A ($23), obteniéndose así una dirección de 16 bits ($C03C). Este byte y el siguiente ($C03D) serán tratados como un puntero a la dirección efectiva ($F84B) cuyo contenido se carga finalmente en A. Dado que X se suma a A antes del acceso indirecto, se habla de un direccionamiento indirecto preindexado. La otra alternativa, el postindexado, exige el cálculo de la dirección indirecta previo a la indexación

Este ejemplo muestra el uso general del direccionamiento indexado. Cuando las direcciones a las que se refiere un programa pueden ser cambiadas, es preferible emplear punteros de dirección fija para señalar estas posiciones. De esta manera, los cambios en las posiciones actuales sólo exigen cambios en el contenido de los punteros.

El uso más común de esta técnica consiste en la estructura conocida como *tabla de salto (jump table)*, que no es más que una tabla de punteros. Todo sistema operativo contiene un gran número de valiosas rutinas que realizan las funciones elementales de la máquina. Por ejemplo, la lectura de un carácter del teclado o la visualización de un carácter en la pantalla. Muchos programas en código máquina tendrán que valerse de estas rutinas en uno u otro momento. En muchos casos, se podrá acceder a ellas gracias a una tabla de salto, lo que quiere decir que estas rutinas apuntadas por las ta-

Ian McKinnell

Cortesía de Air Call

Acceso indirecto

A pesar de que el direccionamiento indirecto es fundamental en las operaciones de un ordenador, resulta difícil encontrar ejemplos análogos en la vida real. Algo se parece, sin embargo, al servicio que prestan los listines de teléfono. Si alguien desea llamar a un abonado, no lo llama directamente pues puede hallarse en cualquier lugar. En su lugar, pide a información que le indique la dirección e incluso que le ponga en contacto. En este ejemplo, el servicio de la centralita proporciona un acceso indirecto (o direccionamiento) a sus abonados

blas de salto pueden ser cambiadas o reubicadas en memoria sin que ello implique la modificación de los programas que las utilizan. Cuando se diseña una nueva versión del sistema operativo o se actualiza la ROM, no suelen respetarse las primitivas posiciones de estas rutinas; pero si la tabla de salto permanece en su posición, con los punteros modificados de modo que reflejen las nuevas direcciones, cualquier software escrito para el antiguo sistema operativo puede ejecutarse en el nuevo.

Una técnica muy común que se emplea en muchos sistemas operativos consiste en retener un punto de entrada y realizar todas las llamadas de subrutina a esta única dirección. Uno de los registros de la CPU se suele añadir para proporcionar un código de función, que sirve para determinar qué subrutina es la llamada. Este código es empleado como un índice o desplazamiento dentro del vector correspondiente de la tabla de salto, pasando el control a través de este puntero a la rutina deseada.

Como ejemplo, supongamos que tenemos cuatro Kbytes de ROM, colocados en $F000, de los cuales los primeros 256 bytes (desde el $F000 hasta el $F0FF) contienen una tabla de hasta 128 direcciones de subrutinas almacenadas en la ROM. La rutina de entrada (la dirección a través de la cual se direccionan todas las demás rutinas del SO) se encuentra en $F100 y espera recibir en el acumulador B un código de función con un valor entre 0 y 127. Este código se emplea por la rutina de entrada para pasar el control a las subrutinas apropiadas y devolverlo después al programa que las ha llamado una vez finalizada su ejecución. La rutina de llamada para la función número 1 es:

```
LDB #1      coloca el código de la función en B
JSR $F100   llama a la subrutina de entrada
```

En sí misma, la subrutina de entrada sería:

```
LDX $F000   dirección del inicio de la tabla de salto
LSLB        desplaza B un lugar a la izquierda (left shift: como si
            multiplicara el contenido de B por dos), dado que toda
            entrada en la tabla tiene dos bytes de longitud. De esta
            manera, el puntero apropiado para el código 1 de
            función se almacena en $F002 y $F003, mientras que el
            indicador del código 2 está en $F004 y $F005, etc.
BRA [B,X]   pasa el control a la dirección encontrada en la posición
            número B de la tabla
```

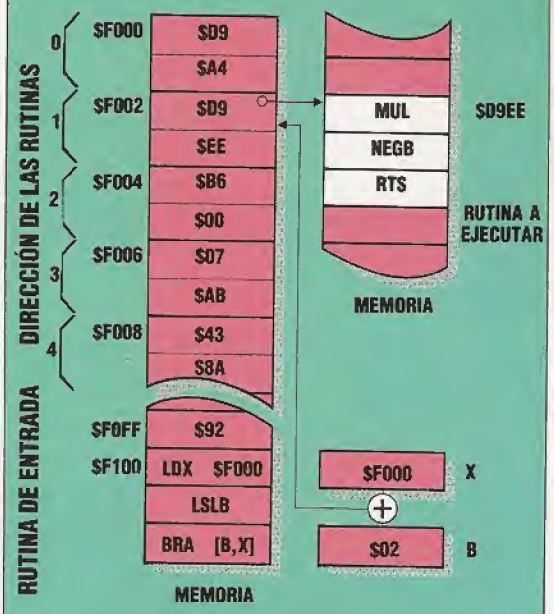
Obsérvese cómo el paso a la rutina se realiza con BRA (o JMP) y no con BSR (o JSR); esto es así para que la RTS, al final de la rutina de SO, devuelva el control directamente al programa desde donde se realizó la llamada en vez de retornar a la rutina de entrada.

Nuestro ejemplo siguiente muestra otro uso del direccionamiento indirecto cuando se trata con una pantalla cuyo mapa de visualización se tiene en memoria; en muchos micros la memoria de pantalla ocupa un sector de la memoria principal y puede accederse a ella directamente en caso de que se requiera mayor velocidad. Supongamos, simplificando, que la pantalla ocupa un bloque de memoria que va desde el $E000 hasta el $E3FF, representando 16 líneas de 64 caracteres. La posición del cursor es un valor de 16 bits dentro de esos límites y se encuentra en $E400. La primera subrutina borra la pantalla escribiendo el carácter de espacio (código 32 en ASCII) sobre cada posición de los 16×64 caracteres. La segunda subrutina va a imprimir el carácter contenido en A sobre la pantalla y en el lugar que ocupa en este momento el cursor, a menos que el carácter sea precisamente el de "re-

torno de carro" (13 en ASCII), en cuyo caso borrará lo que queda de línea y posicionará el cursor al comienzo de la línea siguiente. Éste se representa por el carácter subrayado ("—").

```
SPACE   EQU 32          Código ASCII de espacio
CR      EQU 13          Código ASCII de retorno de carro
HOME    EQU $E000       Comienzo de la memoria de pantalla
LENGTH  EQU 1024        Tamaño de la memoria de pantalla
                        (16 líneas×64 caracteres=1024)
CURSOR  EQU $E400       $E400 y $E401 juntos apuntan a la
                        posición actual del cursor en el área
                        de memoria de pantalla

        ORG $1000
CURCHR  FCB 95          Carácter subrayado (ASCII 95)
***Subrutina para borrar la pantalla***
        LDA #SPACE      Carácter espacio en A
        LDX #HOME       Sitúa el cursor al inicio de la pantalla
                        Almacena en CURSOR la posición
        STX CURSOR      actual de éste (o sea, $E400,$E401)
        LDB #LENGTH     Introduce en B el tamaño de la
                        pantalla
LOOP1   STA [CURSOR]    Deposita un espacio en la posición
                        actual del cursor
        INC CURSOR      Incrementa la posición del cursor
        DECB            Decremento de la cantidad de
                        memoria que queda entre la posición
                        del cursor y el final de la memoria
        BGT LOOP1       Espacio siguiente hasta agotar toda
                        la memoria de pantalla
        STX CURSOR      El cursor vuelve a su posición inicial
        LDA CURSOR      Introduce el carácter del cursor en A
                        (código ASCII)
        STA [CURSOR]    Almacena el carácter del cursor en la
                        posición actual de éste
        RTS
***Subrutina para visualizar el carácter guardado en A, si es
visible***
        CMPA SPACE      El espacio en blanco es el primer
                        carácter imprimible en ASCII
        BLT NOTP        Si el acumulador contiene un valor
                        ASCII menor que 32, no se puede
                        visualizar: GOTO NOTP
        STA [CURSOR]    Almacena la posición actual del
                        cursor
        INC CURSOR      Incrementa la posición del cursor
CHKEOS  LDX #HOME       Comprueba el final de la pantalla
        LEAY #LENGTH,X  Fin de pantalla introducido en Y
        CMPY CURSOR     Si la posición del cursor excede el
                        final de la pantalla entonces...
        BGT FINISH      se alcanzó el final de la pantalla, por
                        lo tanto GOTO FINISH
***Subrutina para desplazar la pantalla***
SCROLL  LEAY 64,X       Y es la longitud de una línea tomada
                        de X (final de la memoria de pantalla)
        LDB #LENGTH     Calcula la cantidad a desplazar
        SUBB #64        Resta 64 a la longitud
LOOP 2  LDA,Y+          Mueve los caracteres una línea hacia
                        atrás (nótese el autoincremento,
                        véase p. 1098)
        STA,X+
        DECB
        BGT LOOP2       Bucle hasta acabar el desplazamiento
        LDD CURSOR      Cursor en el comienzo de la última
                        línea
        SUBD #64
        STD CURSOR
        BRA FINISH
***Subrutina para comprobar si hay "retorno de carro"***
NOTP    CMPA #CR        Este carácter no visualizable es el de
                        "retorno de carro"?
        BNE FINISH      Si no es así, ignorarlo
        LDD CURSOR      Se puede investigar cómo esto
        ANDB            obtiene el inicio de la línea siguiente
        #%11100000      (basta analizar la máscara AND)
        ADDD #64
        STD CURSOR
        BRA CHKEOS      Comprueba el final de la pantalla
FINISH  LDA CURSOR      Se introduce en A el carácter del
                        cursor
        STA [CURSOR]    Lo almacena en la posición actual del
        RTS             cursor
```

La tabla de salto



La tabla de salto de este ejemplo consiste en una lista de 128 punteros de dirección de dos bytes, situados entre $F000 y $F0FF. Cada uno de estos punteros contiene la dirección de inicio de una rutina situada, a su vez, en cualquier sitio de la memoria. Para ejecutar cualquiera de estas rutinas necesitamos tan sólo cargar el acumulador B con el código de la función ($01, p. ej.) que sirve para identificar la rutina (en este ejemplo, situada en $D9EE) y después se realiza un salto JSR hacia la denominada "rutina de entrada", empezando aquí en la dirección $F0FF. Supondremos que estas rutinas están en ROM (dado que forman parte del software situado en ROM, tal como el sistema operativo), de manera que podemos consultar el código de la función y la dirección de inicio de la rutina de entrada en el manual del programador.
La rutina de entrada multiplica el código de la función por dos y lo emplea como desplazamiento respecto a la dirección de inicio de la tabla para encontrar el puntero deseado de la dirección de la rutina: el indicador de la rutina $01 está colocado, por ejemplo, en $F002 (=$F000+2×$01), el puntero de la rutina $02 está en $F004 (=$F000+2×$02), y así sucesivamente. El puntero es entonces empleado por la rutina de entrada en una instrucción de bifurcación indirecta para pasar el control a la rutina de turno colocada en $D9EE. Obsérvese que la rutina de entrada bifurca (más que llama) a la rutina de ejecución, de modo que cuando se encuentra RTS, el control se devuelve al punto del programa en que la rutina de entrada fue llamada por primera vez.
La ventaja de una tabla de salto está en que permite a los programadores volver a diseñar y a colocar las rutinas direccionadas por la tabla, y que los programas escritos antes de estas revisiones pueden ser ejecutados en el nuevo sistema.

El origen de las especies

Atari ha producido versiones del conocido juego "Pacman" (Comecocos) para numerosos micros. Veamos cómo funciona en el Spectrum y el Vic-20

Pacman (Comecocos) fue el juego recreativo de laberinto original, y estableció la fórmula que desde entonces ha seguido el software para ordenadores personales como *Atic Atac* y *Jet Set Willy*. En tales juegos el personaje central debe viajar a través de una red de corredores o habitaciones, recogiendo, mientras tanto, tesoros y sorteando los diversos peligros que lo acechan durante su camino. Bajo ciertas condiciones, es posible que el jugador invierta los papeles de los monstruos atacantes; en *Sabre Wulf*, por ejemplo, tropezar con ciertas orquídeas hace que el jugador sea invisible para las criaturas de la selva. Esta idea se tomó prestada de los juegos de aventuras, en los cuales la posesión de una espada mágica o un objeto similar coloca al jugador en una condición ventajosa.

En *Pacman*, el personaje que da nombre al juego se coloca en el centro de un laberinto, y el jugador debe intentar guiar esta figura alrededor de la pantalla mientras va tragándose los puntos que iluminan el camino. Se debe mantener el ojo atento a los fantasmas perseguidores, quienes exhiben una tozuda determinación a atrapar a Pacman en las diversas calles sin salida que abundan en el laberinto. La ingestión de una "píldora de energía" permite al jugador perseguir y comerse a los fantasmas, consiguiendo de este modo puntos extras, y unos bocados de frutas que van apareciendo al azar contribuyen, si se los come, a aumentar el marcador.

Establecer comparaciones entre distintas versiones del mismo juego podría parecer injusto; al fin y al cabo, un juego diseñado para una determinada máquina obviamente aprovechará mejor las capacidades de ese micro. Esto es así en especial para los juegos diseñados específicamente para las máquinas Atari, que disfrutan de merecida fama, aunque son caros. No obstante, cabe decir que las nuevas versiones de *Pacman* no están a la altura de la versión original.

En los ordenadores personales Atari, *Pacman* configura un enorme laberinto con excelentes gráficos y sonido, movimiento uniforme de los sprites y varios niveles de destreza, y salvando el hecho de que el juego no se ejecute en una máquina exclusiva, esta versión es una réplica fidedigna del juego recreativo. Las versiones para el Spectrum y el Commodore no están en el mismo nivel. En el caso del Vic-20, los programadores se enfrentaron al problema habitual de tener que hacer caber un cuarto de litro en una jarra de un octavo de litro: el laberinto es de sólo una cuarta parte del tamaño de su equivalente Atari, mientras que los sprites son el doble de grandes de lo que eran originalmente, y en este espacio tan restringido se hace difícil esquivar a los fantasmas. No obstante, los gráficos están bien definidos, el movimiento es uniforme y el sonido es tan bueno como el de la versión Atari. En la versión para el Spectrum se proporcionan instrucciones más completas y se le ofrece al jugador la opción de control por teclado en vez de utilizar las palancas de mando necesarias para las versiones Atari y Commodore. Es muy poco lo que los programadores de Atarisoft podrían haber hecho con las limitadas facilidades de sonido del Spectrum, pero los gráficos son sumamente decepcionantes; a pesar de que el laberinto es bastante parecido al del original, el inestable movimiento hace que jugar al *Pacman* sea algo parecido a mirar una película de cine mudo. Las limitaciones de la versión para el Vic se explican en virtud de las restricciones propias del hardware, pero resulta difícil entender por qué la versión para el Spectrum es tan pobre.

Cuando el *Pacman* hizo su primera aparición, en 1980, se convirtió muy rápidamente en una sensación, pero en la actualidad parece un poco anticuado. Hace apenas unos años, Atarisoft podría haber vendido miles de ejemplares de *Pacman*, pero hoy en día existen en el mercado muchos juegos recreativos mejores.

Pacman: Para todos los ordenadores Atari, el Commodore Vic-20 y el Spectrum
Editado y distribuido por: Atari, AUDELEC, Compás de la Victoria, 3, 29012 Málaga, España
Autores: Atari
Palanca de mando: Necesaria
Formato: Atari y Vic-20, cartucho; Spectrum, cassette

Liz Dixon

Por la senda correcta

Una vez visto cómo se puede mover un robot, vamos a analizar las formas de controlar su desplazamiento

De laberintos y ratones

Ratón...
Las competiciones de Micro Mouse, en las cuales compiten ratones robots en el recorrido de un laberinto, han sido una valiosa fuente de conocimiento práctico y experiencia técnica para muchos aficionados a los robots. El Quester de David Buckley, que vemos en la fotografía, lleva una amplia gama de sensores (ópticos, sonoros y sensibles al tacto)

Y laberinto
Cada microrratón tiene un período de práctica en el cual "aprende" el trazado del laberinto por cualquier método que no requiera comunicación externa, y luego debe recorrerlo contra reloj, siendo el objetivo básico llegar al centro del laberinto en el menor tiempo posible

Marcus Willy

El procedimiento más sencillo para desplazar un robot implica la utilización de un dispositivo mecánico que "lee" una ficha de forma especial insertada en aquél. El contorno de la ficha es seguido por un pequeño detector que a su vez opera una serie de palancas para controlar la dirección del ingenio electrónico. En el pasado se podían adquirir coches en miniatura y pequeños robots de juguete que funcionaban de esta forma. La ficha que contenía el programa se creaba utilizando unas tijeras para cortar. El autómata se movía de acuerdo al contorno del borde recortado.

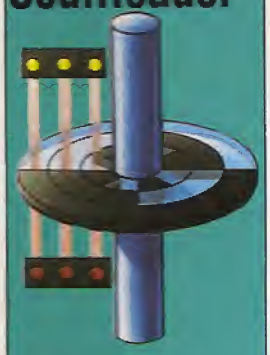
Otros robots empleaban dispositivos que les permitían seguir un recorrido establecido mediante relés electromecánicos internos. Sin embargo, la aplicación de estos procedimientos mecánicos para el control del movimiento era limitada, por la sencilla razón de que los componentes mecánicos suelen ser caros y relativamente inexactos. Pero sí proporcionan un precedente para los métodos actuales.

Uno de los mejores procedimientos empleados hoy en día se basa en hacer que el robot siga una pista especialmente trazada para él en el suelo. Esto es similar al método que utilizan los juegos de coches de carreras, que poseen una patilla guía insertada en una ranura continua de una pista de carreras a escala. Las dos formas más corrientes de autómatas cibernéticos que siguen una pista son, no obstante, aquellos que se desplazan a través de una línea dibujada en el suelo y los que se guían por medio de un cable.

En los robots que siguen una línea se utiliza un sensor de luz, normalmente una célula fotoeléctrica o un sensor infrarrojo, para determinar si el robot está situado sobre una zona clara o una zona oscura. Si el color del suelo es oscuro y el de la línea es claro, la salida del sensor siempre estará en su nivel más alto cuando el sensor esté directamente sobre la línea. Por consiguiente, si el robot sigue siempre la ruta que proporciona la salida eléctrica más elevada del sensor, en ningún momento se apartará de la línea trazada.

Esta técnica plantea un problema: ¿qué hace el robot cuando la salida del sensor cae, indicando que ha abandonado la línea? Con un sistema de un único sensor, lo mejor que puede hacer el robot es dar vueltas en derredor hasta que la señal de salida del sensor vuelva a subir, indicando que se halla nuevamente sobre la línea. Luego puede continuar en la dirección en la que esté encarado. Este sistema no es tan aleatorio como podría parecer. Por ejemplo, si el robot estuviera yendo hacia la iz-

Codificador



Un codificador de eje permite que un robot sepa cuánto han rotado los ejes de sus ruedas. El dispositivo es un disco calibrado que se coloca en un eje. El plato circular está dividido en cierto número de círculos concéntricos, marcados cada uno en zonas que son opacas o transparentes. Con una fuente luminosa y un sensor de luz para cada anillo se puede calcular la orientación exacta del eje.
La precisión de un codificador de eje depende del número de anillos de que disponga el disco. En nuestra ilustración vemos un codificador de eje de tres anillos, que puede codificar los números binarios del 000 al 111 (decimales del 0 al 7). La precisión de este codificador es de 360/8=45 grados. Ocho anillos concéntricos darían una precisión de 1,41 grados

quierda cuando la salida del sensor cayera, lo lógico entonces es que gire hacia la derecha para volver a encontrar la línea. Asimismo, habiendo hallado ésta es lógico que suponga que la dirección en la cual debe dirigirse ahora esté en algún punto entre el recorrido que estaba siguiendo cuando perdió la línea (izquierda) y el recorrido que ha debido realizar para hallarla otra vez (derecha).

Un sistema que reduce el tiempo que tiene que perder un robot "extraviado" tratando de hallar otra vez la dirección utiliza dos sensores orientados a cada uno de los lados de la línea. Esto significa que cuando el robot está sobre la línea, la salida de ambos sensores es baja. Si el robot comienza a apartarse de ésta, la salida de uno de los sensores se eleva. Ello significa que el robot sabe inmediatamente que se ha desviado y en qué dirección ha cometido el error. Si se apartara hacia la derecha, se elevaría la salida del sensor izquierdo y tomaría esto como una señal para girar hacia la izquierda, lo que lo devolvería nuevamente a su recorrido.

Este sistema no requiere tener una línea blanca sobre un fondo negro, sino que puede funcionar igualmente bien con una línea oscura sobre un fondo claro. Lo importante es el contraste y que la programación le diga al robot qué hacer cuando un sensor lee un valor incorrecto.

El otro sistema que se utiliza para los robots que siguen huellas implica enviar una pequeña corriente eléctrica a lo largo de un cable colocado sobre el suelo. Esta corriente genera un pequeño campo magnético alrededor del cable, que es detectado por el sensor. Éste no necesita ser un sensor complicado: una pequeña bobina de alambre captará el campo magnético y producirá un pequeño voltaje que se puede después ampliar y que actuará exactamente de la misma manera en que lo hacen los sensores. Con frecuencia, los robots industriales que necesitan desplazarse se basan en un cable enterrado en el suelo debajo de ellos. Si dependieran de una línea pintada sobre la superficie, todo iría bien hasta que el suelo se ensuciara.

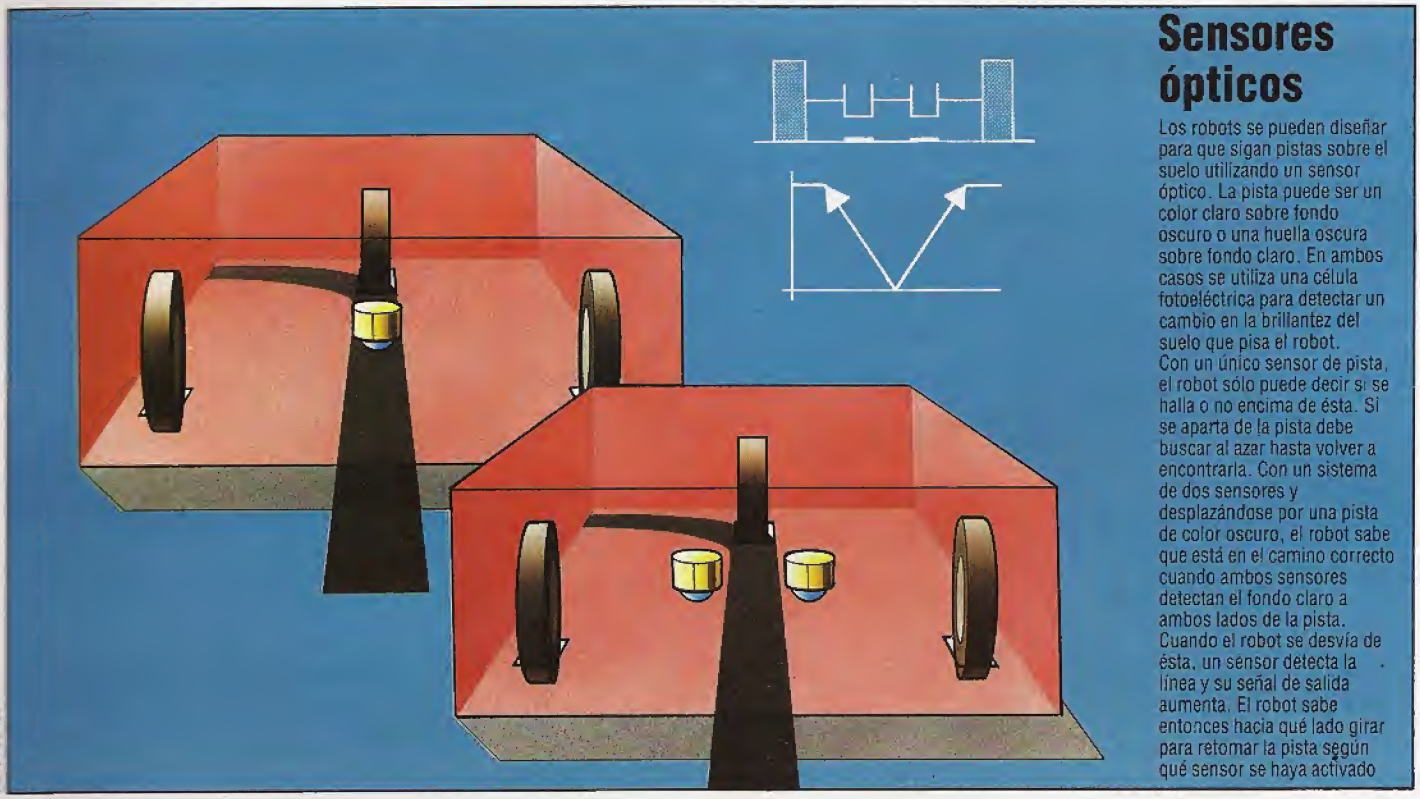
Control remoto

Otro método es el de que un operador humano controle al robot a distancia. Esto es particularmente útil cuando las tareas a llevar a cabo por el robot podría realizarlas igualmente un ser humano, pero en un entorno demasiado hostil y poco seguro. Ejemplos de ello son la desactivación de bombas, la manipulación de materiales radiactivos o materiales químicos peligrosos y el trabajo en zonas demasiado calientes, frías o peligrosas para un ser humano.

Un robot de esta clase muy conocido es el soviético Lunojod I, depositado en la superficie lunar por el Luma 16 en 1970. Se trataba de un robot sobre ruedas que recogió información sobre la superficie del satélite bajo el control por radio de científicos situados en la Tierra.

El control de robots de este tipo es algo diferente del que se ejerce sobre un avión a escala por medio de la radio. La señal emitida por ésta puede ser analógica, cuya intensidad varía a tenor del movimiento que se requiera del robot, o digital, que compone un patrón de bits que ofrece detalles relativos a los movimientos a efectuar. Las comunicaciones analógicas tienden a ser menos eficaces que los métodos digitales, porque hay diversos factores que podrían interferir en la intensidad de la señal de una transmisión analógica. Pruebe a escuchar una emisora de radio lejana y observe cómo varía la recepción según la hora del día y las condiciones climatológicas. El mismo tipo de problemas puede afectar a las señales utilizadas para comunicaciones con robots.

Los métodos digitales también pueden plantear



Sensores ópticos

Los robots se pueden diseñar para que sigan pistas sobre el suelo utilizando un sensor óptico. La pista puede ser un color claro sobre fondo oscuro o una huella oscura sobre fondo claro. En ambos casos se utiliza una célula fotoeléctrica para detectar un cambio en la brillantez del suelo que pisa el robot.
Con un único sensor de pista, el robot sólo puede decir si se halla o no encima de ésta. Si se aparta de la pista debe buscar al azar hasta volver a encontrarla. Con un sistema de dos sensores y desplazándose por una pista de color oscuro, el robot sabe que está en el camino correcto cuando ambos sensores detectan el fondo claro a ambos lados de la pista. Cuando el robot se desvía de ésta, un sensor detecta la línea y su señal de salida aumenta. El robot sabe entonces hacia qué lado girar para retomar la pista según qué sensor se haya activado

Kevin Jones

problemas, en especial cuando la interferencia hace que se pierdan bits o que se los inserte incorrectamente. Para evitar esto, los mensajes a los robots se suelen repetir varias veces. Estos autómatas cibernéticos sólo se ponen en acción luego de haber recibido reiteradamente un mensaje enviado en idénticos términos.

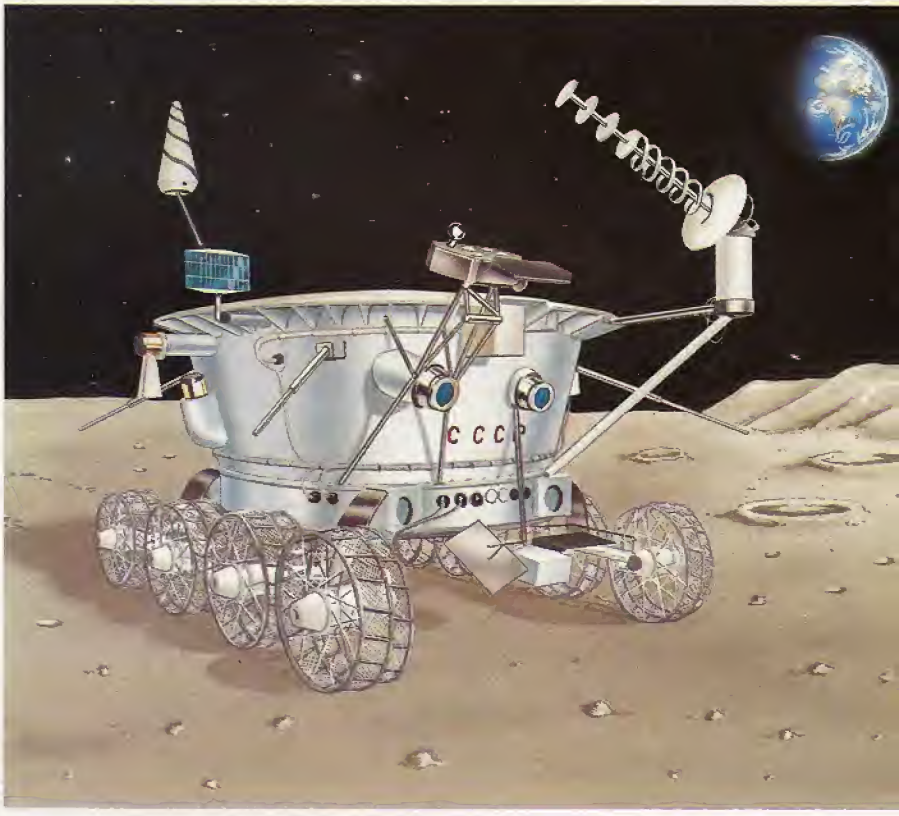
Sistemas de realimentación

Una técnica más sofisticada es la utilización de un sistema de "bucle", en el cual el robot le proporciona al transmisor un feedback (retorno, realimentación) relativo a la señal que acaba de recibir. Esto se podría considerar como un diálogo entre el transmisor y el robot. Por ejemplo, el transmisor puede decir "hacia adelante" y, habiendo recibido el mensaje, el robot le dirá "¿has dicho hacia adelante?", a lo que el transmisor respondería "sí", y el robot se movería entonces de acuerdo a la indicación. Esto podría ayudar a evitar errores graves si el robot estuviera manipulando residuos nucleares o estuviera a punto de caerse dentro de un cráter de la Luna.

Las mismas técnicas generales se pueden aplicar a otros medios de control remoto. Por ejemplo, algunos robots se pueden controlar a través de emisores infrarrojos de la clase que se utiliza para los dispositivos de control a distancia para aparatos de televisión. O se podrían controlar mediante sonido ultrasónico, algo así como un silbato para perros, o mediante sonido audible de una naturaleza distintiva, tales como una serie de palmadas. Sea cual sea el método que se utilice, las técnicas que se aplican para transmitir el mensaje y asegurar que el robot lo haya recibido son las mismas.

Si el operador humano se hallara bastante cerca del robot, tal vez no fuera necesario emplear técnicas tan sofisticadas: al robot se le podrían transmitir las órdenes a través de un cable de conexión. También existe la posibilidad de utilizar más de un cable, lo que equivaldría a tener varios canales en el caso de un avión controlado por radio. Pero, en el caso del robot, los cables adicionales generalmente se emplean para proporcionar una comunicación en paralelo en vez de en serie (se envía una serie de bits en paralelo a lo largo de los cables en vez de enviarse como una serie de impulsos a lo largo de un cable). Esto permite una comunicación más rápida con el robot. Quizás aún más importante sea el hecho de que la mayoría de los ordenadores disponen de una puerta en paralelo. Ésta comunica de forma excelente las instrucciones al robot desde el teclado de un ordenador.

Si el movimiento del robot lo ha de controlar un operador humano sentado frente al teclado de un ordenador y puede ver al robot, entonces en principio hay muy poca diferencia entre controlar al robot a través de un operador humano y hacerlo por medio de un ordenador. Ello se debe a que, al igual que el avión controlado por radio, el operador siempre puede ver lo que está haciendo el robot y corregir de inmediato cualquier error. Pero si el autómata cibernético se halla a cierta distancia (en la Luna, p. ej., e incluso en la habitación contigua), o si se ha de controlar a través de un programa dentro del ordenador en vez de a través de instrucciones de teclado en tiempo real, entonces el robot ha de ser ligeramente más inteligente.



Steve Cross

Un salto de gigante para los robots
El Lunojod I fue depositado en la superficie lunar por la URSS en 1970 para recoger información acerca de la naturaleza de la superficie y la atmósfera del satélite terrestre. No era un auténtico robot (se lo controlaba por radio desde la Tierra), pero su disposición invulnerable e invariable frente a las rigurosas condiciones lunares permitió que la nave espacial que lo llevó pudiera regresar con una aportación científica superior de la que hubiera sido posible con astronautas y sus imprescindibles y elaborados sistemas de acondicionamiento. Al igual que todos los objetos del espacio controlados a distancia, el Lunojod experimentaba un intervalo de demora de tres segundos entre su transmisión de información a la Tierra y la recepción de una señal de control en respuesta

Esencialmente, lo que se necesita es alguna forma de realimentación (*feedback*). Esto es un proceso que permite que el sistema ajuste lo que está haciendo en relación a lo que ya ha efectuado y a lo que debería estar realizando. Por ejemplo, si desea que un robot recorra tres metros por una habitación y lo está controlando directamente, puede empezar a moverlo, evaluar su progreso y detenerlo cuando lleve recorridos tres metros. Esto es así porque usted tiene una realimentación visual con el robot: ve hasta dónde ha avanzado, hasta qué distancia desea que vaya, y puede corregir sus acciones en consecuencia.

En ausencia de la realimentación sensorial humana, el robot ha de disponer de alguna propia si es que debe desplazarse con precisión. El robot que sigue una línea utiliza la realimentación de la línea que está siguiendo en el suelo y, del mismo modo, el robot controlado por ordenador debe emplear alguna realimentación si es que ha de recorrer exactamente tres metros hacia adelante. Uno de los métodos más comúnmente aplicados para proporcionar la realimentación necesaria es un *codificador de eje*, que consiste en un disco circular fijado a los ejes principales de las ruedas del robot y que proporciona una medida muy exacta de cuánto han rotado. Por lo tanto, si el ordenador le envía instrucciones al robot indicándole que avance tres metros, el robot puede empezar a moverse y, al mismo tiempo, controlar las señales provenientes de sus codificadores de eje para ver hasta dónde se ha desplazado. Si el robot debe avanzar más, puede continuar desplazándose. Cuando llegue allí se puede detener, y si por cualquier motivo no acaba en el lugar correcto, entonces siempre se puede apoyar en la cantidad correcta calculada a partir de la información enviada por los codificadores de eje para corregir su error.

Sinfonía en software

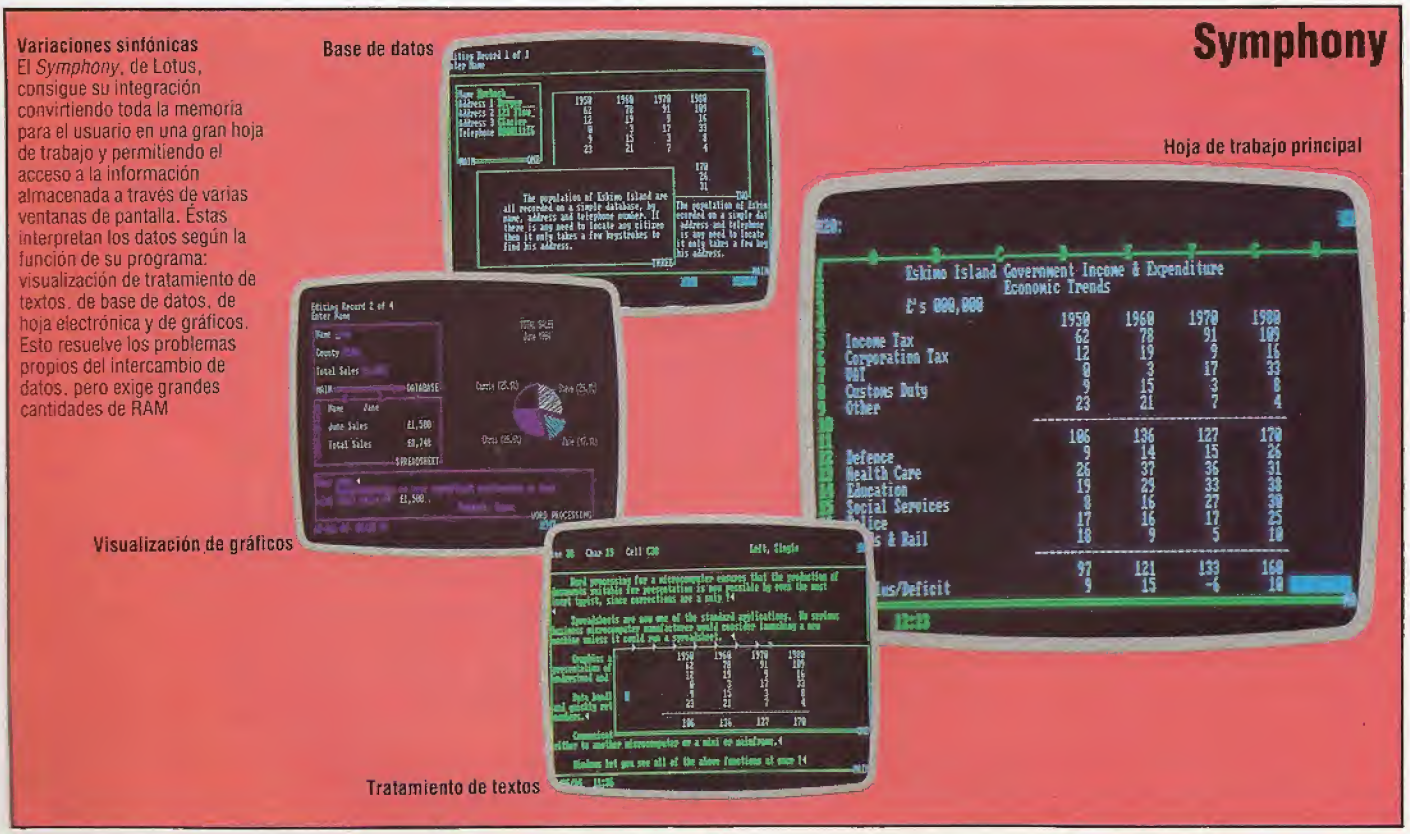
Examinemos tres paquetes de software integrado, cuyas técnicas se aplicarán pronto a máquinas más asequibles

Como ya hemos visto, el software integrado exige un entorno en el cual el usuario tenga acceso instantáneo a la totalidad de las diversas tareas que pueda requerir, donde los procedimientos de operatoria permanezcan constantes con independencia de la aplicación que se esté utilizando, y donde la información se pueda transferir libremente entre distintas aplicaciones. Existen muchas formas diferentes de conseguir estos objetivos.

El *Lotus 1-2-3* emplea el familiar formato de hoja electrónica, en el que las cifras y las fórmulas se entran en una cuadrícula de "celdas" y se pueden corregir libremente y volver a calcular al instante. No obstante, ofrece muchas facilidades adicionales y puede ser usado para mucho más que para la predicción y el análisis financiero. Las celdas de la hoja electrónica se pueden utilizar para almacenar información tal como nombres y números de teléfono además de datos numéricos, de modo que se podría emplear un área específica de la cuadrícula como una tabla conteniendo detalles que vengan al caso; por ejemplo, una lista de clientes y sus correspondientes números de cuenta bancaria. Dado que el *1-2-3* ofrece funciones de búsqueda y de reorganización para tal información, esta zona de la cuadrícula se puede utilizar en realidad como una pequeña base de datos. También es posible tomar un conjunto de celdas que contengan datos numéricos y emplear este paquete para visualizar esta información en forma de distintos tipos de gráficos, eliminando por consiguiente la necesidad de un programa separado de gráficos para gestión. Por último, las capacidades del *1-2-3* para tratamiento de textos permiten que se lo pueda utilizar para escribir informes, si bien las limitaciones de memoria impiden su empleo como un auténtico procesador.

Esta combinación de distintas facilidades conduce a que el *1-2-3* sea suficiente para los requerimientos de muchos usuarios. Debido a que toda la información para distintas aplicaciones está contenida en una única hoja electrónica, es fácil obtener resultados que serían imposibles con programas tradicionales. Por ejemplo, supongamos que un usuario de este paquete gestiona varios quioscos de periódicos situados en distintos puntos de una gran ciudad y necesita registrar cifras de ventas semanales, mensuales, trimestrales y anuales para cada puesto. La mejor forma de hacerlo consiste en colocar la ubicación de cada quiosco y sus cifras de ventas en una hoja electrónica. Las fórmulas se escriben de modo tal que las únicas cifras que el usuario debe modificar son los recibos semanales para cada puesto: las otras cifras se ajustan luego automáticamente.

Variaciones sinfónicas
El *Symphony*, de Lotus, consigue su integración convirtiendo toda la memoria para el usuario en una gran hoja de trabajo y permitiendo el acceso a la información almacenada a través de varias ventanas de pantalla. Éstas interpretan los datos según la función de su programa: visualización de tratamiento de textos, de base de datos, de hoja electrónica y de gráficos. Esto resuelve los problemas propios del intercambio de datos, pero exige grandes cantidades de RAM



Hasta ahora todo esto es material para una hoja electrónica estándar, pero ¿y si el usuario deseara colocar los puestos por orden de venta, de modo que el emplazamiento con mayores ventas fuera el primero en la lista? Estos puestos inicialmente se entrarían por orden alfabético, pero sería necesario reclasificarlos cada semana al recibir las nuevas cifras de ventas. Con el *Lotus 1-2-3* esto se puede hacer fácilmente y con rapidez. El propietario de los quioscos de periódicos podría desear un gráfico semanal que reflejara el rendimiento de cada uno; una secuencia de pulsaciones de teclas permitiría recuperar esta información para la hoja electrónica/base de datos, visualizarla en forma de gráfico e imprimirla.

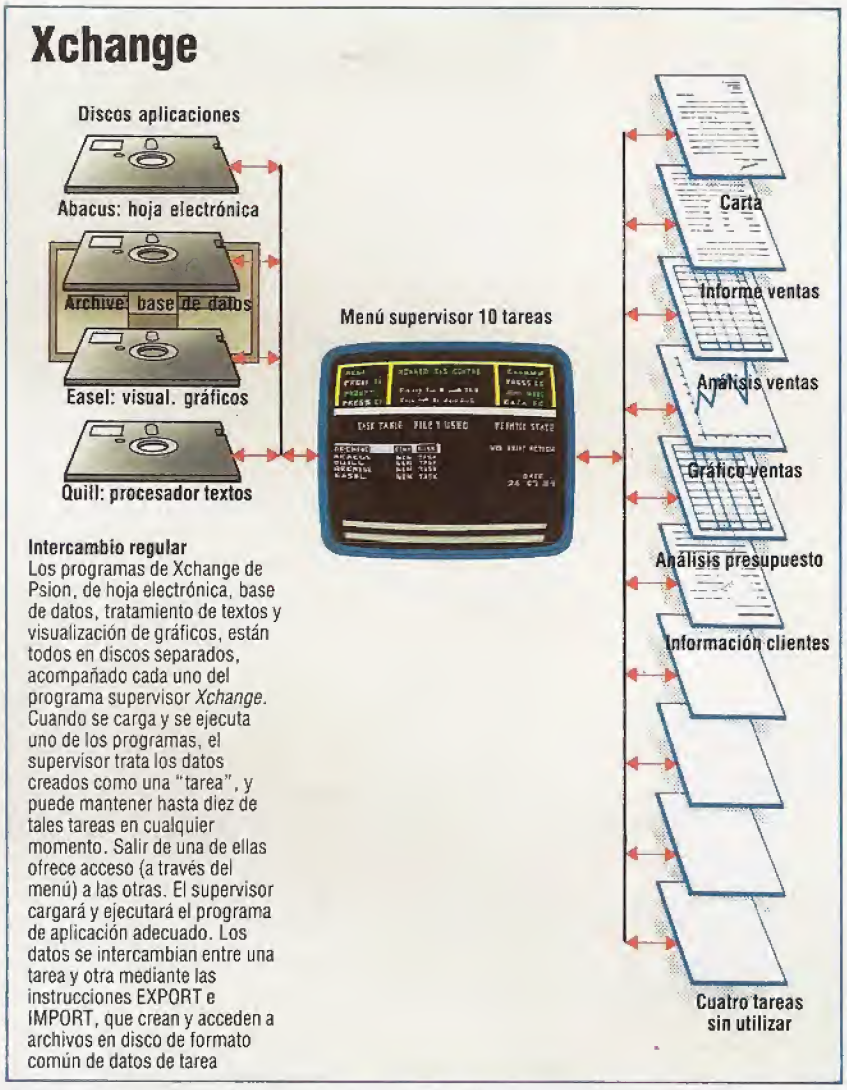
Lotus Symphony

Lotus ha continuado el concepto del *1-2-3* en el *Symphony*, que sigue el mismo principio de basar las aplicaciones en el formato de hoja electrónica. Sin embargo, el *Symphony* permite que el usuario divida la visualización en pantalla en ventanas separadas, cada una de las cuales se centra en una parte diferente de la hoja. Cada ventana está formateada de acuerdo a la información que visualiza.

Si la información a visualizar está en formato de texto, la ventana asume la forma de una pequeña pantalla para tratamiento de textos, con márgenes y posiciones de tabulación claramente marcados. Si se requiere una visualización de gráficos, la ventana muestra los ejes etiquetados y escalados. La información de base de datos se visualiza con una pantalla propia para cada entrada; esta pantalla se parece a una tarjeta de fichero. Por tanto, aunque el *Symphony* es realmente una hoja electrónica muy mejorada, da la impresión de tener cuatro grandes aplicaciones, todas en pantalla y simultáneas.

Al igual que el *1-2-3*, el *Symphony* puede "aprender" determinadas secuencias de pulsaciones de teclas, de modo que el usuario tiene la posibilidad de automatizar cualquier operación que se lleve a cabo con frecuencia. Los pequeños programas que activan la secuencia se denominan *macros de teclado*. El *Symphony* incluye asimismo su propio lenguaje de programación de alto nivel. Los programas se almacenan en la hoja de trabajo de la misma forma que todos los otros datos, y tienen acceso a todas las operaciones disponibles: por lo tanto, si el usuario debe efectuar tareas como facturación o control de stocks, puede escribir el programa en el lenguaje de programación del paquete y automáticamente pasará a formar parte de todas las aplicaciones del "entorno" *Symphony*. Una vez se haya familiarizado con el *Symphony*, le resultará más fácil escribir programas en su lenguaje propio que utilizar un lenguaje de programación independiente como el BASIC, porque el paquete ya se ocupa de ciertas tareas como dibujar gráficos o buscar y organizar datos.

El *Symphony* no es más que uno de los diversos sistemas similares que se encuentran en el mercado. El *Framework*, de Ashton Tate, es un serio competidor: proporciona una gama similar de funciones pero oculta sus estructuras de datos subyacentes en un grado aún mayor. Tanto el *Symphony* como el *Framework* son caros y exigen grandes cantidades de memoria. El primero funciona con 320 Kbytes de RAM, pero en realidad exige 512 Kbytes para aprovechar al máximo sus facilidades, mientras que el segundo necesita un mínimo de 256 Kbytes. A consecuencia de estos requerimientos, los paquetes sólo operarán en micros de 16 bits.



Kevin Jones

Es interesante resaltar que tanto el *Symphony* como el *Framework* funcionan con todo el paquete cargado en memoria, no requiriendo intercambio de información entre el disco y la memoria, como sucede con la mayoría de los programas de gestión. En teoría, por supuesto, la memoria de ordenador continúa volviéndose cada vez más barata, de modo que no deja de ser comprensible que quienes desarrollan software den por sentado que la mayoría de los usuarios dispondrán de ella en grandes cantidades. En la práctica, sin embargo, éste no es el caso todavía y pasará algún tiempo antes de que esta integración, que consume tanta memoria, se convierta en un lugar común. A pesar de que un programa como el *Symphony* establece nuevos estándares de rendimiento, esta clase de software aún está restringido por limitaciones de hardware; el *Symphony* consigue cumplir tan eficientemente las labores asignadas únicamente gracias a que se trata de un programa muy amplio y cuidadosamente acabado.

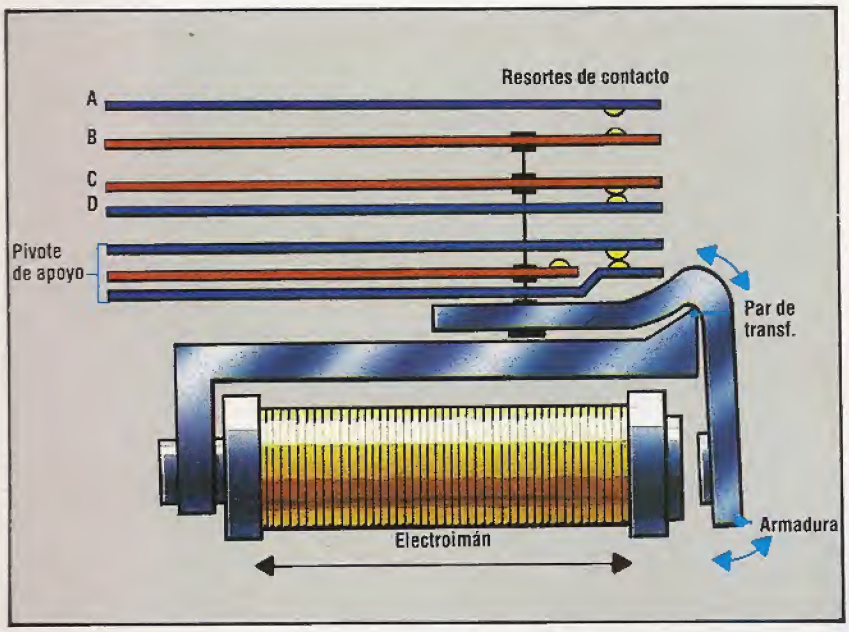
En los últimos veinte años se ha desarrollado un método alternativo para proporcionar software integrado y en la actualidad están comenzando a salir al mercado paquetes con este procedimiento.

Fuente de poder

Continuamos nuestra serie de "Bricolaje" con una explicación de cómo construir una caja de relés

Utilizando una caja de relés, un ordenador es capaz de encender y apagar las luces de una vivienda a intervalos preestablecidos. Estaría, también, en condiciones de programar una grabadora de video o de audio para que, en ausencia del usuario, le grabara una determinada emisión de televisión o radio.

Los relés eléctricos son interruptores on/off que se pueden activar mediante una señal eléctrica. En nuestra aplicación, los relés se emplean para conmutar aparatos de corriente y alto voltaje utilizando una señal de corriente y bajo voltaje. Existen muchos tipos de relés, pero el más común es el de tipo armadura, que se basa en un solenoide para abrir y cerrar conexiones.



El relé abre y cierra los contactos por la acción de pequeños movimientos de la armadura. Un voltaje adecuado aplicado a la bobina solenoide genera un campo magnético que atrae a la armadura. Cuando ésta oscila hacia la bobina, los contactos de resorte fijados en el otro extremo de la armadura se mueven verticalmente hacia arriba.

La disposición de la ilustración está en la posición "no activada", es decir, sin ningún voltaje aplicado al solenoide. En esta posición, el par de contacto AB está abierto y el par CD, cerrado. Cuando se activa el solenoide, los resortes B y C se mueven hacia arriba, haciendo que A y B se cierren y que C y D se abran. Esta disposición se podría utilizar en una de dos formas: para encender un circuito mientras se apaga el otro, o bien, más simplemente, para cerrar o abrir un circuito.

Además de esta modalidad de operación, un relé puede actuar como un mecanismo de transferencia. En el diagrama, los tres resortes inferiores están dispuestos de modo tal (en la posición no activada) que los resortes superior e inferior están en contacto. Cuando se activa el solenoide, el resorte del medio se mueve hacia arriba y hace contacto con el resorte superior, rompiendo por tanto el contacto entre los resortes superior e inferior.

Lista de componentes

Cantidad	Artículo
2	Relé 8-15 voltios de contacto de 10A 240V
2	Conector de red individual
2	Base de 29 mm
2 metros	Cable de red 3 núcleos 6 amperios
2	Enchufe de red
*	Enchufe 4 mm
*	Cable plano 2 vías 0,5 m
*	Pequeños trozos de veroboard

Nota: Los artículos señalados con un * le deben de haber sobrado de otros proyectos (véase p. 1004). Estas cantidades proporcionarán dos cajas de relés de conector único. La caja de salida puede activar cuatro de tales cajas (si así se deseara éstas podrían ser dobles) o incluso cajas de triple conector; los principios de construcción son exactamente los mismos.

Dentro de la caja

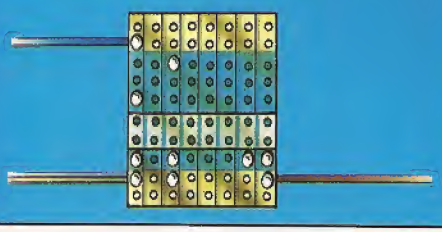
Verifique todas las conexiones para asegurar la seguridad y la continuidad, e inspeccione la placa otra vez por si hubiera puentes entre pistas. Asegúrese de que no haya comunicación entre el cable de la red y las líneas de señal.
Pegue la placa en una esquina de la base utilizando una cola de resina epoxia. Algunas colas domésticas de uso general conducen la electricidad, de modo que evítelas. Si no estuviera seguro acerca de las propiedades conductoras del adhesivo elegido, haga una prueba: coloque una delgada franja de cola sobre un trozo de cartón, déjela secar y luego conecte un tester a ambos extremos; si el medidor marca, ¡utilice otra cola! Una vez seca la cola, atornille la tapa en la caja y coloque los enchufes de 4 mm en las líneas de señal (éstas pueden ser del mismo color, porque el relé trabajará independientemente de la dirección de la corriente). Ahora conecte el enchufe de 13 A al cable de la red. El relé tiene una potencia nominal de 10 A, pero ponga en el enchufe un fusible de sólo 5 A: éste permite que se controlen los aparatos de hasta 1,2 Kw.

¡Atención!

Éste es un proyecto muy sencillo, pero con la electricidad debe ponerse sumo cuidado.
- **Desconecte todas las fuentes de alimentación antes de trabajar con cualquier componente.**
- **Compruebe conexiones y aislamientos con un tester antes de aplicar potencia por primera vez.**
- **Evite saltarse o abreviar etapas. Recuerde que ¡LA POTENCIA ELÉCTRICA PUEDE SER MORTAL!**

La placa de circuitos

Corte la placa según muestra la ilustración, de modo que quepa cómodamente en una esquina de la caja base. Efectúe los cortes de pistas y suéldela al relé tal como indica el diagrama.
Antes de seguir adelante verifique con suma atención la placa. Utilice el tester para comprobar posibles puentes entre pistas: ¡un error podría ser mortal!
Suelde el cable de la red marrón y el cable plano de dos vías en su lugar en el tablero. Quite una de las ranuras preformadas de la base para aceptar los cables; pero haga un nudo en ellos antes de pasarlos a través; el nudo evitará que un tirón accidental de los cables dañe la placa. Suelde a ésta un trocito de conductor de red aislado y conéctelo al terminal de tornillo "cargado" del conector. Conecte los cables azul y amarillo-verde de la red a los terminales neutral y a tierra, respectivamente



Programa de prueba

Después de construida la caja de relés y comprobadas todas las conexiones, podemos probarla escribiendo un programa para encender y apagar un dispositivo que funcione con la corriente de la red. Éste podría ser una lámpara de mesa. Ésta se debe enchufar al conector de fuente de alimentación de la caja de relés, conectando los cables de señal a los terminales positivo y negativo de la línea 0 de la caja de salida de bajo voltaje. Los cables de señal se pueden conectar en cualquiera de los terminales sin afectar el funcionamiento del relé. El cable de la red de la caja del relé se conecta, entonces, en un enchufe de la pared. Después de hechas las conexiones, entre este programa: encienda la lámpara durante cinco segundos y después la apaga.

```
10 REM PRUEBA RELE DE LA RED
20 RDD=&FE62:REGDAT=&FE60
30 ?RDD=255:REM TODAS SALIDA
40 ?REGDAT=1:REM ENCENDER LA LUZ
50 TIEMPO=0: REM ESTABLECER TEMPORIZADOR
60 REPEAT
70 UNTIL TIEMPO>500
80 ?REGDAT=0:REM APAGAR LA LUZ
```

```
10 REM PRUEBA RELE DE LA RED CBM 64
20 RDD=56579:REGDAT=56577
30 POKE RDD,255:REM TODAS SALIDA
40 POKE REGDAT,1:REM ENCENDER LA LUZ
50 T=TI:REM ESTABLECER TEMPORIZADOR
60 IF (TI-T)<300 THEN 60
70 POKE REGDAT,0:REM APAGAR LA LUZ
```

Si después de ejecutar el programa la lámpara no se encendiera, desenchufe la caja de relés de la red antes de comprobar las conexiones.

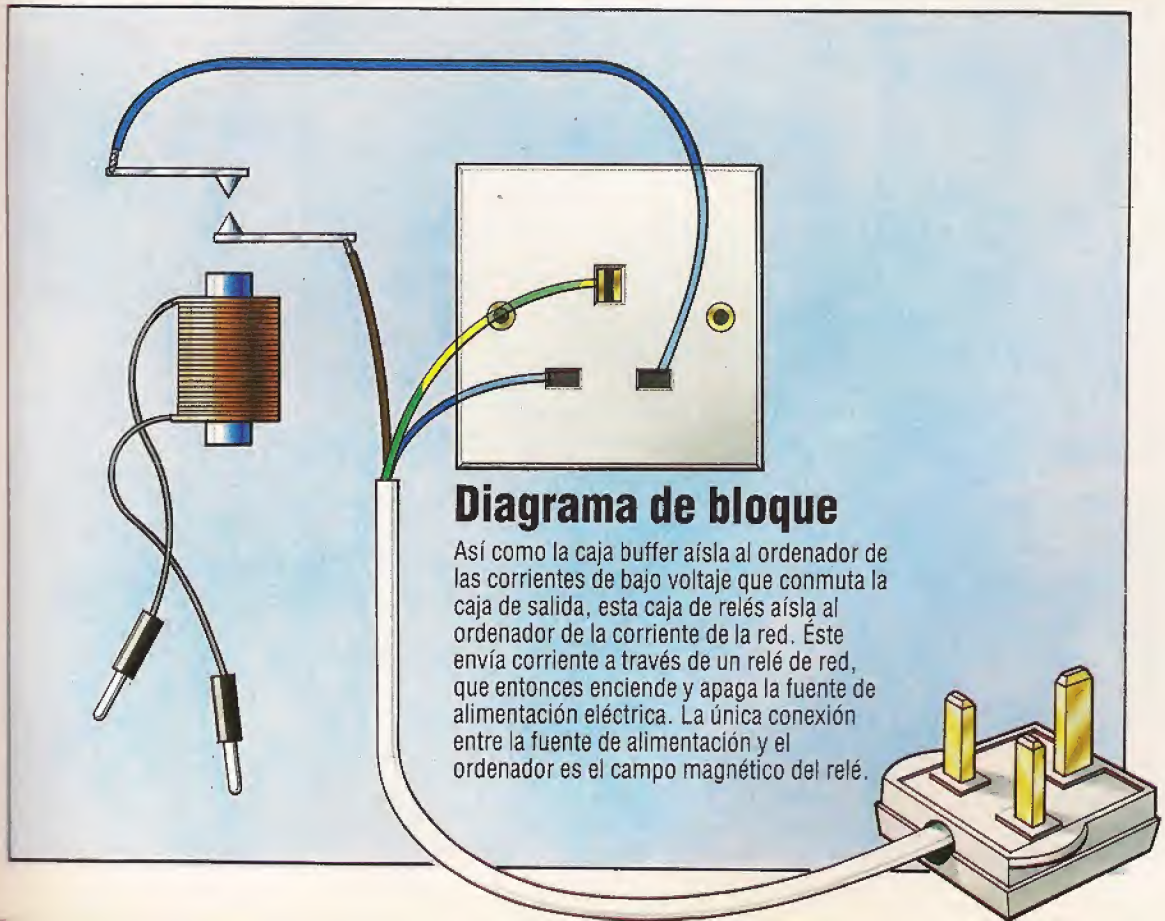


Diagrama de bloque

Así como la caja buffer aísla al ordenador de las corrientes de bajo voltaje que conmuta la caja de salida, esta caja de relés aísla al ordenador de la corriente de la red. Éste envía corriente a través de un relé de red, que entonces enciende y apaga la fuente de alimentación eléctrica. La única conexión entre la fuente de alimentación y el ordenador es el campo magnético del relé.

Proyecto de código Morse

```
 10 REM ****************************************
 11 REM*          PRACTICA MORSE CBM 64        *
 12 REM*        ENCHUFE UNA LAMPARA EN         *
 13 REM*              EL RELE                  *
 14 REM*        ENTRE CUALQUIER SERIE          *
 15 REM*        ALFA, Y SE REPRODUCIRA         *
 16 REM*          EN MORSE CON BEEP Y          *
 17 REM*            LUZ INTERMITENTE           *
 20 REM ****************************************
100 GOSUB 2000:                          REM INICIO
150 FOR L=0 TO 1 STEP 0
200 PRINT"ENTRE SU MENSAJE"
220 PRINT"DIGITE 'ADIOS' PARA SALIR"
240 INPUT"MENSAJE";M$$
300 ML=LEN(M$$):M$=""
320 FOR K=1 TO ML
330 C$=MID$(M$$,K,1)
340 IF C$=>"A"ANDC$<="Z" THEN M$=M$+C$
350 IF C$=" " THEN M$=M$+C$
360 NEXT K: IF M$="" THEN NEXT L
400 ML=LEN(M$)
420 FOR K=1 TO ML
440 CH$=MID$(M$,K,1):CH=ASC(CH$)-64
450 IF CH=-32 THEN FORPP=1TO6*DE:NEXT PP
460 IF CH<>-32 THEN GOSUB 3000
480 FOR PP=1 TO 3*DE:NEXT PP
500 NEXT K
550 IF M$="ADIOS" THEN L=1
600 NEXT L
900 END
2000 REM ***************** INICIAL *****************
2100 DIM M$(26)
2110 RDD=56579:REGDAT=56577:POKE RDD,255
2120 DE=25:RX=2*DE
2130 V=54296:LF=54272:HF=54273:W=54276
2140 A=54277:S=54278
2150 FOR K=LF TO LF+24:POKEK,0:NEXT K
2160 POKEA,24:POKES,129:POKEV,15
2200 DATA ".-","-...","-.-.","-..","."
2220 DATA "..-.","--.","....","..","
2240 DATA ".---","-.-","
2260 DATA "-.","---","
2280 DATA "
2300 DATA ".--","-..-","-.--","--.."
2400 FOR K=1 TO 26: READ M$(K):NEXT K
2990 RETURN
3000 REM ************* FLASH & BEEP *************
3050 PRINT CH$,M$(CH)
3100 N=LEN(M$(CH))
3200 FOR C=1 TO N
3220 D=DE-(ASC(MID$(M$(CH),C,1))-46)*RX
3240 POKE REGDAT,1                :REM FLASH
3250 POKE LF,172:POKE HF,57:REM BEEP
3260 POKE W,33:FOR PP=1 TO D:NEXT PP
3270 POKE W,32
3280 POKE LF,0:POKE HF,0          :REM QUITAR BEEP
3290 POKE REGDAT,0                :REM QUITAR FLASH
3300 FOR PP=1 TO DE:NEXT PP
3320 NEXT C
3490 RETURN
```

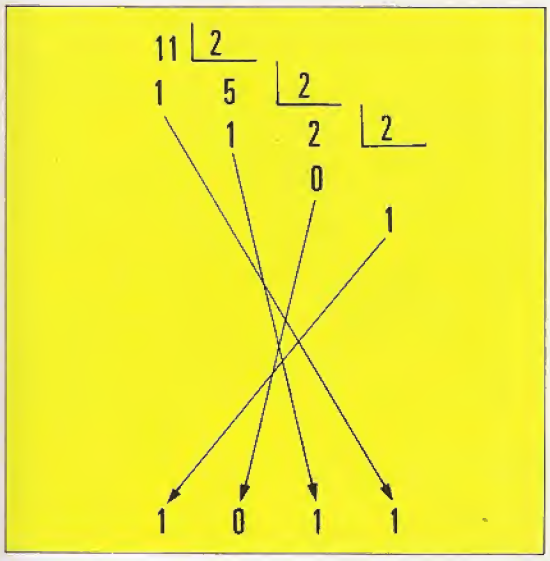
```
 10 REM ****************************************
 11 REM*          PRACTICA MORSE BBC           *
 12 REM*        ENCHUFE UNA LAMPARA EN         *
 13 REM*            EL RELE DE RED:            *
 14 REM*        ENTRE CUALQUIER SERIE          *
 15 REM*        ALFA, Y SE REPRODUCIRA         *
 16 REM*          EN MORSE CON BEEP Y          *
 17 REM*            LUZ INTERMITENTE           *
 20 REM ****************************************
100 PROCinicializar
120 FIN=FALSE
150 REPEAT
200 PRINT"ENTRE SU MENSAJE"
220 PRINT"TECLEE 'ADIOS' PARA SALIR"
240 INPUT"MENSAJE",MS$
300 ML=LEN(MS$):M$=""
320 FOR K=1 TO ML
330 C$=MID$(MS$,K,1)
340 IF C$>="A" AND C$<="Z" THEN M$=M$+C$
350 IF C$=" " THEN M$=M$+C$
360 NEXT K:IF M$="" THEN UNTIL FALSE
400 ML=LEN(M$)
420 FOR K=1 TO ML
440 CH$=MID$(M$,K,1):CH=ASC(CH$)-64
450 IF CH=-32 THEN PROCdemora(6*DE*IX)
460 IF CH<>-32 THEN PROCbeepflash
480 PROCdemora(3*DE)
500 NEXT K
550 IF M$='ADIOS' THEN FIN=TRUE
600 UNTIL FIN
900 END
2000 REM ***************** INICIO *****************
2050 DEF PROCinicializar
2100 DIM M$(26)
2110 RDD=&FE62:REGDAT=&FE60:?RDD=255
2120 DE=3:RX=2*DE:IX=30
2200 DATA ".-","-...","-.-.","-..","."
2220 DATA "..-.","--.","....","..","
2240 DATA ".---","-.-","
2260 DATA "-.","---","
2280 DATA "
2300 DATA ".--","-..-","-.--","--.."
2400 FOR K=1 TO 26:READ M$(K):NEXT K
2990 ENDPROC
3000 REM ************* FLASH & BEEP *************
3020 DEFPROCbeepflash
3050 PRINT CH$,M$(CH)
3100 N=LEN(M$(CH))
3200 FOR C=1 TO N
3220 D=DE-(ASC(MID$(M$(CH),C,1))-46)*RX
3240 ?REGDAT=1                    :REM FLASH
3260 SOUND 1,-15,200,D            :REM BEEP
3270 PROCdemora(IX*D)
3290 ?REGDAT=0                    :REM QUITAR FLASH
3300 PROCdemora(DE*IX)
3320 NEXT C
3490 ENDPROC
4000 REM *************** DEMORA ***************
4100 DEFPROCdemora(tiempo)
4200 FOR DD=1 TO tiempo:NEXT DD
4490 ENDPROC
```

Cód. Morse	
A	·—
B	—···
C	—·—·
D	—··
E	·
F	··—·
G	——·
H	····
I	··
J	·———
K	—·—
L	·—··
M	——
N	—·
O	———
P	·——·
Q	——·—
R	·—·
S	···
T	—
U	··—
V	···—
W	·——
X	—··—
Y	—·——
Z	——··
.	·—·—·—
,	——··——

De decimal a binario

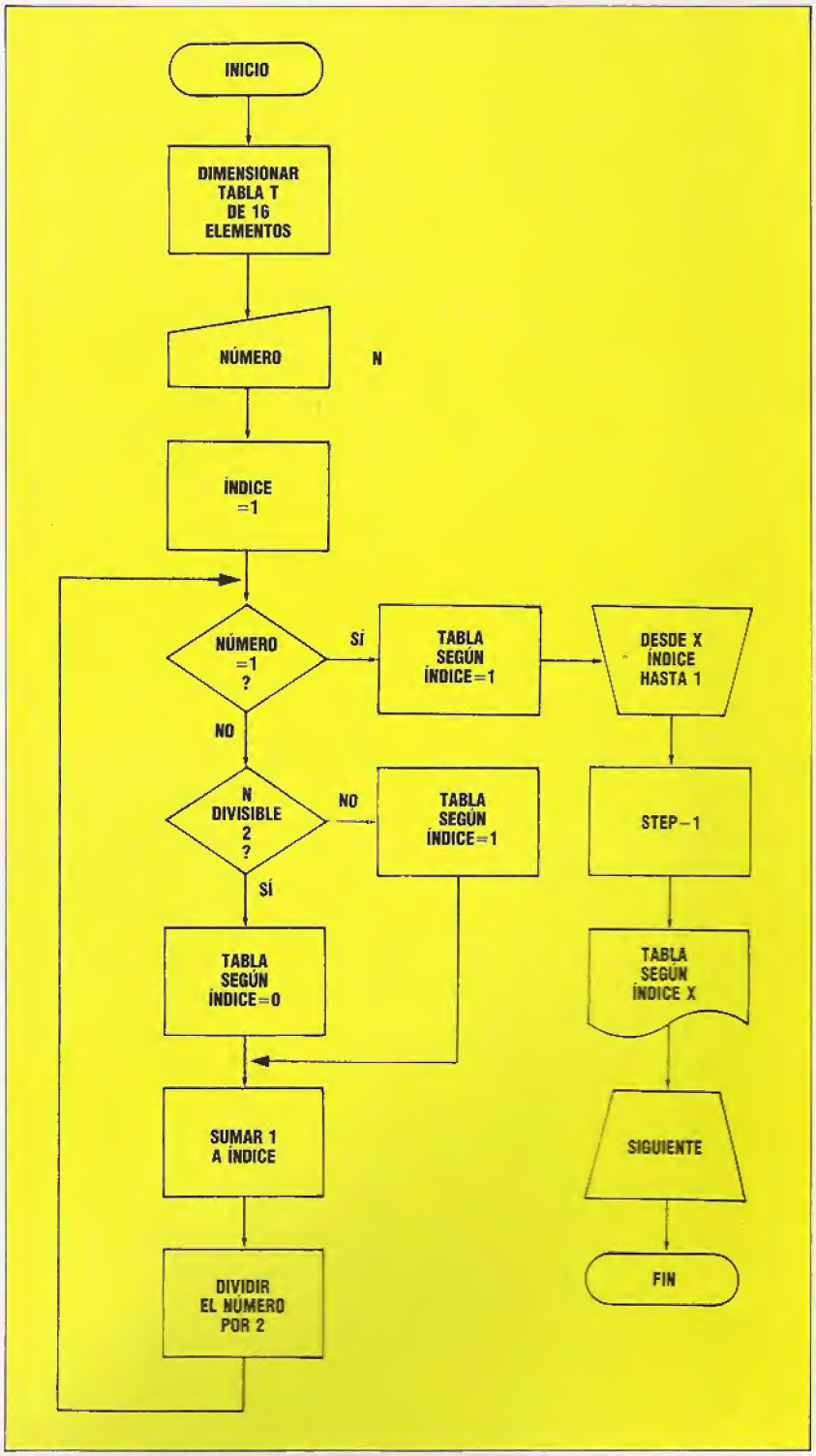
En este último capítulo de este apartado desarrollaremos un diseño que permite convertir una cantidad decimal en su correspondiente número binario

Para realizar la operación de convertir un número decimal en binario se va dividiendo la cantidad decimal entre la base del sistema a que se quiere convertir (en este caso, binario=base 2), hasta que el dividendo ya no contenga al divisor, es decir, hasta que no se pueda dividir más por la base. Entonces se toma el último cociente y todos los restos, en orden inverso a como han aparecido, o sea de derecha a izquierda en orden ascendente. Veamos un ejemplo: $11_{(10}=1011_{(2}$



De este modo, se ve la necesidad de guardar cada uno de los restos que surjan. Para ello se dejarán en una tabla que hemos creado con capacidad para 16 elementos; así podrán representarse números con un valor máximo de 65 535.

```
  5 REM CONVERSION BINARIA
 10 DIM T(20)
 20 INPUT "ENTRAR VALOR DECIMAL";N
 30 E=1
 40 IF N=1 THEN GOTO 100
 50 IF N=INT(N/2)*2 THEN T(E)=0 : GOTO 70
 60 T(E)=1
 70 E=E+1
 80 N=INT(N/2)
 90 GOTO 40
100 T(E)=1
110 FOR X=E TO 1 STEP-1
120 PRINT T(E);
130 NEXT X
140 END
```



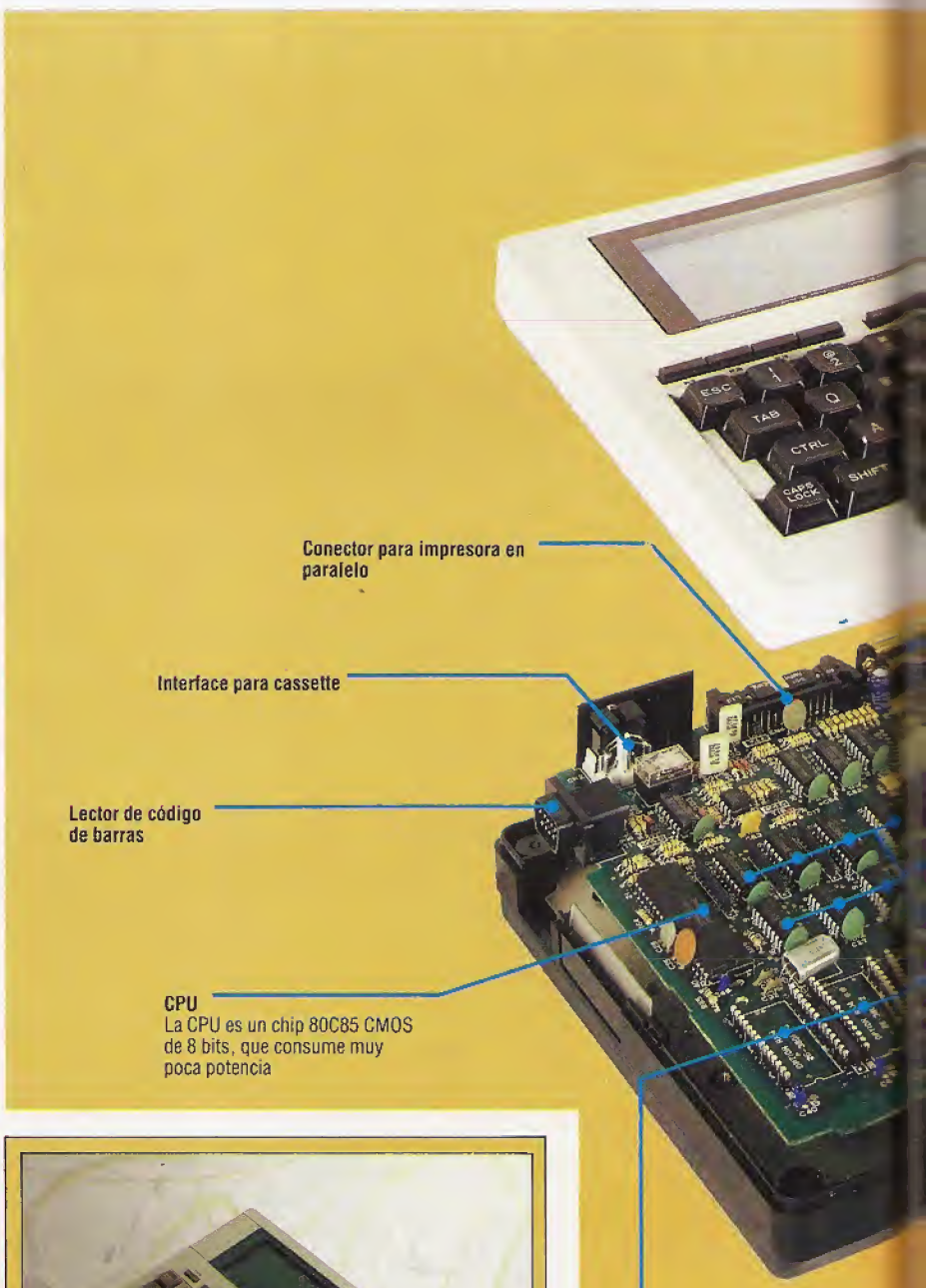
Un trío de calidad

Demos una mirada a una máquina "de regazo", el Tandy Modelo 100, y comparémosla con dos dignos competidores

El proceso por el cual un fabricante adquiere un producto acabado, modifica algunos elementos para darle un aspecto exclusivo y lo empaqueta después como un artículo "fabricado para el cliente", se conoce como *ingeniería del distintivo*. En el campo de la electrónica de consumo, esta técnica ya existe desde hace tiempo, con productos tales como televisores y equipos de alta fidelidad. La misma técnica se está comenzando a aplicar ahora en el mercado de ordenadores, y tres populares máquinas portátiles (el Tandy Modelo 100, el NEC PC8201A y el Olivetti M10) son producto de un acuerdo de este tipo. La firma japonesa Kyocera es la que fabrica los tres ordenadores y se los vende a Tandy, NEC y Olivetti, que ponen su propia carcasa a las máquinas y las comercializan bajo su respectivo nombre. En este capítulo analizaremos el Tandy 100 y pondremos de relieve las diferencias entre esta máquina y sus hermanas.

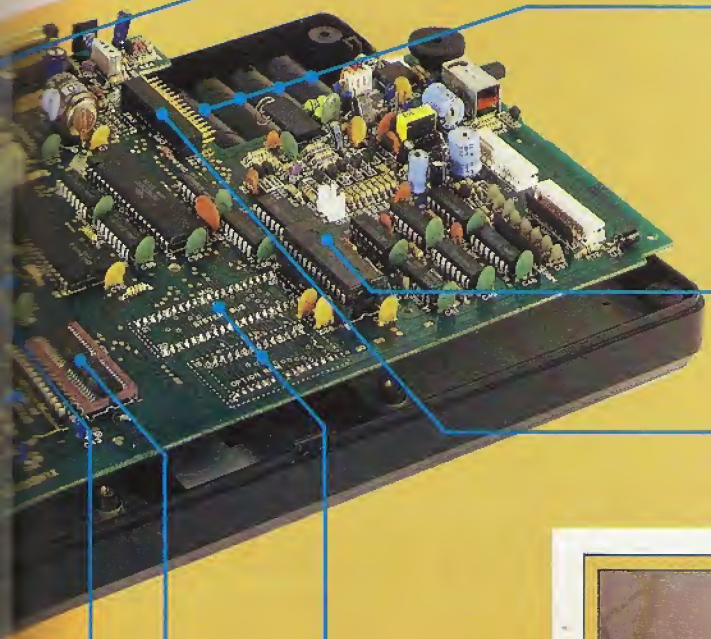
Pesando poco menos de 1,8 kg, los modelos Tandy, NEC y Olivetti se inscriben claramente en la categoría de máquinas "de regazo". El Modelo 100 tiene un teclado estilo QWERTY completo, software incorporado basado en ROM y una pantalla en cristal líquido (LCD) a pilas. Puede funcionar completamente con la energía de éstas, y el contenido de la RAM no se pierde cuando se apaga la máquina. Los archivos se pueden almacenar en RAM y acceder directamente como si la memoria estuviera en cassette o en disco. El Modelo 100 también se puede conectar a una cassette o unidad de disco para almacenamiento externo, pero la memoria permanente hace que almacenar datos importantes "sobre la marcha" resulte sencillo.

La pantalla LCD proporciona ocho líneas de 40 caracteres y ofrece la posibilidad de mezclar texto y gráficos. La visualización se compone de 15 360 puntos, cada uno de los cuales se puede referenciar individualmente. Los caracteres se forman en una matriz de 6 por 8 y se pueden visualizar caracteres en mayúscula y en minúscula. El Modelo 100 incorpora un juego completo de caracteres internacionales, así como otro especial de caracteres para gráficos, a diferencia de la máquina NEC, que sólo posee tres caracteres para gráficos. Tanto el NEC como el Tandy tienen pantallas LCD que yacen extendidas sobre sus carcasas, pero el Olivetti M10 configura una pantalla móvil que se puede inclinar para obtener un ángulo de trabajo cómodo, proporcionando, por consiguiente, mayor flexibilidad. El NEC y el Tandy tienen mandos de contraste



NEC PC8201A
Si bien el NEC PC8201A es exactamente del mismo tamaño que sus hermanos, posee un teclado claramente diferente. Las teclas del cursor se han trasladado para conformar un pequeño racimo, las teclas de función se han reducido de 8 a 5, y el trazado del teclado es ligeramente distinto. Además, el NEC sólo posee tres de los programas estándares en ROM: *Text*, *Schedule* y *Telecom*

Conector para modem
Ésta es una puerta RS232 estándar para comunicaciones en serie. El software para telecomunicaciones está incorporado en la máquina



Fuente de alimentación
El Modelo 100 funciona hasta 20 horas con 4 pilas alcalinas "AA". La memoria interna se conserva hasta 30 días en virtud de pilas recargables de níquel-cadmio, que se recargan automáticamente cuando está conectada a la red

ROM estándar
Este chip contiene el BASIC Microsoft y el software incorporados

Conector LCD
Un cable plano conecta la pantalla LCD con la placa del sistema

Bus del sistema y ranuras de ROM
Estas ranuras vacías son para futura ampliación de la ROM del sistema y control de entrada-salida

RAM de 8 K estándar

Unidad de teclado
Estos chips controlan la entrada y la salida del teclado y contienen los juegos de caracteres

Olivetti M10
La versión Olivetti de esta máquina posee un interesante detalle exclusivo de ella: la pantalla LCD se puede inclinar hasta un ángulo de unos 40°, lo que hace que la visualización resulte más fácil de leer. Posee el mismo teclado que el Tandy 100 y los cinco paquetes de software estándares basados en ROM

TANDY MODELO 100

DIMENSIONES
300 x 215 x 50 mm

CPU
CMOS 80C85 de ocho bits, 2,4 MHz

MEMORIA
RAM de 8 K o 24 K, ampliable en bloques de 8 K hasta un total de 32 K; 32 K de ROM, incluyendo software y BASIC Microsoft

PANTALLA
LCD de 40 columnas x 8 líneas; gráficos de 240 x 64 puntos direccionables; caracteres ASCII e internacionales, 39 caracteres para gráficos

INTERFACES
Impresora en paralelo, cassette, puerta en serie RS232, lector de código de barras, bus del sistema

LENGUAJES DISPONIBLES
BASIC Microsoft

TECLADO
Teclado estándar tipo máquina de escribir, de 56 teclas; teclado numérico incorporado; 8 teclas de función programable; 4 teclas de instrucciones y 4 teclas de control de cursor

DOCUMENTACION
Guía de referencia rápida al BASIC de 48 páginas; detallado manual de operatoria de 200 páginas

VENTAJAS
El Tandy 100 es pequeño y, sin embargo, ofrece la mayoría de las características necesarias para la informática 'seria'. La memoria permanente y el funcionamiento a pilas lo hacen muy portátil

DESVENTAJAS
Las limitaciones de la RAM reducen el uso práctico de la máquina a aplicaciones portátiles. No tendría capacidad para 'superar' su condición de portátil y funcionar como un genuino ordenador de sobremesa

DIFERENCIAS
Olivetti M10: Teclado de 57 teclas; 47 caracteres para gráficos; unidad de pantalla inclinable; un manual para el usuario
NEC PC8201A: 57 teclas; rosa de cursor; 5 teclas de función; 16 K de RAM ampliables a 96 K; sólo tres caracteres para gráficos

Cortesía de CL Projects Ian McKinnell

De tal palo tal astilla
A pesar de ser distribuidos por tres empresas diferentes, el Tandy Modelo 100, el Olivetti M10 y el NECP PC8201A son todos fabricados por la misma empresa, la Kyocera japonesa. Existen algunas pequeñas diferencias entre las tres máquinas, pero hasta un ojo inexperto puede ver que los tres ordenadores portátiles comparten una herencia común

Ian McKinnell

ajustables para conferirle más claridad a la pantalla.

El excelente teclado del Tandy posee teclas especiales para acceder a los gráficos incorporados o para cambiar varias de las teclas alfabéticas convirtiéndolas en un teclado numérico. Utilizando esta facilidad, la tecla M se convierte en 0; J, K y L se transforman en 1, 2 y 3; por su parte, U, I y O se convierten en 4, 5 y 6; finalmente, 7, 8 y 9 conservan su función normal. Las tres máquinas poseen cuatro teclas para el cursor, pero la posición de las mismas varía. Los modelos Tandy y Olivetti poseen cuatro pequeñas teclas, una junto a la otra, situadas arriba y a la derecha del teclado; el NEC PC8201A posee un teclado de cursor, con las cuatro teclas para el cursor formando un cuadrado.

Las máquinas incorporan, asimismo, teclas de función programable, que se utilizan con el software incorporado para dirigir las funciones de tratamiento de archivos y el movimiento en y entre los programas retenidos en ROM. También existen algunas diferencias en este aspecto. El Modelo 100 de Tandy tiene ocho teclas de función, más cuatro teclas adicionales que se utilizan para llevar a cabo tareas internas. PASTE se emplea para trasladar datos de un programa a otro; LABEL les asigna nombres a las teclas de función de modo que el usuario siempre sabe qué función realiza cada tecla; PRINT envía archivos directamente a la impresora, y la tecla BREAK interrumpe la ejecución del programa. Este trazado se repite en el Olivetti, pero el NEC posee cinco teclas de función, programables para un total de 10 funciones, y una tecla Pause.

El Modelo 100 y el M10 se suministran con 8 o 24 K de RAM, que se pueden ampliar a 32 con la adición de un paquete de RAM interna. El NEC es ligeramente diferente: se suministra con 16 K, pero se puede ampliar internamente a 64 o a 96 K si se utiliza la puerta para ampliación incorporada.

El Modelo 100 viene con BASIC Microsoft y un pequeño sistema de "administración doméstica" que dirige el software interno. Al conectarse a la red, se visualizan los archivos almacenados en memoria, junto con los títulos de los programas internos suministrados.

Los programas proporcionados incluyen *Text*, un pequeño procesador de textos apto para apuntar memorándums o escribir cartas o informes cortos; éste es especialmente adecuado para tomar notas y muy útil para periodistas, estudiantes o usuarios de gestión. *Schedule* es un pequeño programa de base de datos, diseñado específicamente como ayuda en el registro de citas, gastos, "cosas que hacer" y otros recordatorios. Una función de búsqueda incorporada facilita la localización rápida de información. Un tercer programa, llamado *Address*, constituye una pequeña base de datos similar que puede parecer innecesaria, dada la existencia de *Schedule*. Por último, hay un programa de comunicaciones basado en RS232 denominado *Telecom*, que permite conectar el Modelo 100 con un modem para comunicaciones telefónicas: con unas pocas pulsaciones de teclas se pueden enviar o recibir datos desde remotos ordenadores. El NEC PC8201A sólo viene con BASIC, *Text* y *Telecom*.

Las tres máquinas están bien provistas de interfaces, disponiendo cada una de una puerta para comunicaciones RS232, una puerta para impresora en paralelo, interface para cassette y un conector para lector de código de barras. Los modelos Tandy y Olivetti incluyen un bus del sistema, mientras que el NEC agrega a su relación de interfaces dos puertas en serie adicionales.

La utilización de una máquina básica, con ligeras diferencias entre los tres modelos distintos, ha significado que los fabricantes puedan proporcionar productos de alta calidad sin que ninguna de las empresas haya tenido que hacer frente sola a todos los costos de desarrollo. Sólo nos resta resaltar que cualquiera de estos tres ordenadores portátiles ofrece una buena relación calidad-precio.

¿Qué motocicleta?

He aquí las versiones del juego del capítulo anterior para otros dos ordenadores: Commodore 64 y BBC Micro

A diferencia de las versiones de BASIC que utilizan el Spectrum y el BBC, el BASIC del Commodore 64 no posee ninguna instrucción que nos permita activar pixels individuales. En la versión del juego que ofrecemos aquí, utilizamos caracteres en baja resolución para dibujar el sendero de las "motos luminosas". Se emplea un carácter espacio invertido, con POKE código 160: para trazar este carácter en la pantalla tenemos que colocar (POKE) este valor en el mapa de pantalla de la memoria y especificar el color en la posición correspondiente.

Al igual que la versión para el Spectrum, el juego para el Commodore no está estructurado para obtener la máxima velocidad de ejecución. En aquellos puntos en los que la velocidad carezca de importancia se introduce algo de estructuración en forma de llamadas a subrutinas para incrementar el marcador y dotar de intermitencia a la pantalla.

Debido a que el BASIC BBC opera considerablemente más rápido que el BASIC Spectrum o Commodore y permite llamar a módulos estructurados a modo de procedimientos, la versión del juego para el BBC es muy estructurada. La mayoría de las versiones de BASIC permiten la estructuración mediante el empleo de subrutinas, pero ello reduce la velocidad de ejecución, puesto que se debe efectuar una búsqueda cada vez que se las llama. El BASIC BBC toma nota de la posición de un procedimiento cuando el mismo es llamado por primera vez, y la almacena en una tabla de referencia.

Commodore 64

```
10 REM CBM 64
15 POKE53281,0:POKE53280,4:REM COLOR PANT./TABL.
20 SC=1024:CO=55296
25 PRINTCHR$(147):REM LIMPIAR PANTALLA
30 X1=2:Y1=12:X2=37:Y2=12
35 DX=1:DM=-1:DN=0:DY=0
40 PRINTCHR$(19)CHR$(158)"JUGADOR 1:"S1
50 PRINTCHR$(19)TAB(27)"JUGADOR 2:"S2
52 FORI=1TO8:PRINTCHR$(17);:NEXT
54 PRINTTAB(14)"PULSE UNA TECLA"
55 GETJS:IFJS<>""THEN55
56 GETAS:IFAS=""THEN56
57 PRINTTAB(14)CHR$(145)"
60 REM BUCLE PRINCIPAL
70 GETA$
80 IFA$="W"THENDY=-1:DX=0
90 IFA$="X"THENDY=1:DX=0
100 IFA$="A"THENDX=-1:DY=0
110 IFA$="D"THENDX=1:DY=0
120 IFA$="0"THENDN=1:DY=0
130 IFA$="M"THENDN=-1:DM=0
140 IFA$="J"THENDM=-1:DM=0
150 IFA$="L"THENDM=1:DN=0
155 Y1=Y1+DY
156 IFY1<1ORY1>24THENF=0:GOSUB1000:GOTO25
157 X1=X1+DX
158 IFX1<00RX1>39THENF=0:GOSUB1000:GOTO25
160 Y2=Y2+DN
162 IFY2<1ORY2>24THENF=1:GOSUB1000:GOTO25
164 X2=X2+DM
166 IFX2<00RX2>39THENF=1:GOSUB1000:GOTO25
167 P1=X1+40*Y1
168 P2=X2+40*Y2
170 IFPEEK(SC+P1)=160THENF=0:GOSUB1000:GOTO25
180 IFPEEK(SC+P2)=160THENF=1:GOSUB1000:GOTO25
190 POKE SC+P1,160
200 POKE CO+P1,14
210 POKE SC+P2,160
220 POKE CO+P2,5
230 GOTO70:REM RECOMENZAR BUCLE
240 :
1000 REM S/R MARCADOR
1020 IF F=1THENS1=S1+1:GOSUB2000:RETURN
1030 S2=S2+1:GOSUB2000:RETURN
1999 :
2000 REM S/R FLASH PANTALLA
2010 FORJ=1TO10
2020 FORI=0TO15
2030 POKE53281,I
2040 NEXTI,J
2045 POKE53281,0
2050 RETURN
```

BBC Micro

```
10 REM BBC
20 MODE1:moto1=0:moto2=0
30 PROCinicializar
40 PROCtecla
50 PROCborde
60 PROCmarcador
70 PROCtrazar
80 PROCteclado
90 PROCleer—punto
100 GOTO70
110 END
120 DEF PROCborde
130 GCOL0,borde
140 MOVE0,0
150 DRAW1279,0
160 DRAW1279,980
170 DRAW0,980
180 DRAW0,0
190 ENDPROC
200 DEF PROCteclado
210 REM JUGADOR UNO
220 IF INKEY(-51)=-1 THEN dx=4:dy=0
230 IF INKEY(-87)=-1 THEN dm=4:dn=0
240 IF INKEY(-66)=-1 THEN dx=-4:dy=0
250 IF INKEY(-70)=-1 THEN dm=-4:dn=0
260 IF INKEY(-67)=-1 THEN dx=0:dy=-4
270 IF INKEY(-34)=-1 THEN dx=0:dy=4
280 IF INKEY(-55)=-1 THEN dm=0:dn=4
290 IF INKEY(-102)=-1 THEN dm=0:dn=-4
300 x=x+dx:y=y+dy:m=m+dm:n=n+dn
310 ENDPROC
320 DEF PROCinicializar
330 x=100:y=490:m=1179:n=490
340 dx=4:dy=0:dm=-4:dn=0
350 col1=1:col2=2:borde=3
360 ENDPROC
370 DEF PROCleer—punto
380 pixel1=POINT(x,y)
390 pixel2=POINT(m,n)
400 IF pixel1 THEN PROCexplosion(x,y,1)
410 IF pixel2 THEN PROCexplosion(m,n,2)
420 ENDPROC
430 DEF PROCtrazar
440 GCOL0,col1:PLOT69,x,y
450 GCOL0,col2:PLOT69,m,n
460 ENDPROC
470 DEF PROCexplosion(ex,ey,cual)
480 FORI=1TO100
490 MOVEex,ey
500 GCOL0,RND(3)
510 PLOT1,RND(50)-25,RND(50)-25
520 NEXT
530 IF cual=2 THEN moto1=moto1+1
540 IF cual=1 THEN moto2=moto2+1
550 PROCinicializar
560 CLS
570 PROCtecla
580 PROCborde
590 PROCmarcador
600 ENDPROC
610 DEF PROCmarcador
620 PRINTTAB(1,0)"Moto Uno=";moto1
630 PRINTTAB(28,0)"Moto Dos=";moto2
640 ENDPROC
650 DEF PROCtecla
660 PRINTTAB(8,12)"PULSE CUALQUIER TECLA PARA EMP."
670 *FX21
680 A$=GET$
690 PRINTTAB(8,12)"
700 ENDPROC
```

Tortuga voladora

En este capítulo desarrollaremos un sencillo juego en el que la tortuga se pierde en el espacio

En nuestro juego *La tortuga del espacio*, ésta se encuentra perdida en la inmensidad del espacio, a mucha distancia de su base, a la que debe regresar. El juego exigirá que imprimamos varios mensajes en la pantalla.

No es de extrañar que la instrucción necesaria para este fin sea PRINT. Después de impreso el mensaje, se desplaza el cursor hasta el comienzo de la siguiente línea.

Para imprimir una sola palabra, a continuación de PRINT va la palabra propiamente dicha; por consiguiente, PRINT "HOLA imprime en la pantalla la palabra "HOLA". PRINT" se utiliza para imprimir la "palabra nula" (una "palabra" que no tiene caracteres).

El efecto de esta instrucción es, simplemente, imprimir una línea en blanco. Si se ha de imprimir más de una palabra, el texto se encierra entre corchetes para indicar que forma una *lista*:

```
PRINT [SU TIEMPO HA CONCLUIDO]
```

PRINT también se emplea para visualizar el contenido de una variable, de modo que PRINT :MARCADOR tomará el valor retenido en la variable "MARCADOR y lo visualizará. En la misma sentencia PRINT se pueden combinar mensajes y valores de variables encerrando toda la instrucción entre paréntesis, como en este caso:

```
(PRINT [SU MARCADOR FUE] :MARCADOR)
```

PRINT1 se comporta exactamente de la misma manera que PRINT, excepto que en este caso el cursor permanecerá al final del texto impreso y no se desplazará hasta la línea siguiente. Esto se puede demostrar entrando:

```
PRINT1 [COMO SE LLAMA UD?]
```

Operaciones de salida

Las instrucciones de LOGO, como HIDETURTLE o PRINT, hacen que suceda algo: se puede decir que las mismas tienen un efecto sobre la tortuga. Sin embargo, otras primitivas del LOGO (XCOR, p. ej.) no producen ningún efecto sobre la tortuga sino que producen un valor. Este valor se utiliza entonces normalmente como parámetro para una instrucción. Por lo tanto, digitar por ejemplo:

```
PRINT XCOR
```

haría que XCOR proporcionara el valor correspondiente a la coordenada actual *x* de la tortuga en la instrucción PRINT, que visualizaría entonces el resultado. Por consiguiente, si el valor corriente de XCOR fuera 20, PRINT XCOR haría que en la pantalla apareciera el número 20. Si se digitara XCOR sólo, aparecería el mensaje RESULT:20. Éste es en realidad un mensaje de error (las versiones LCSI imprimirán YOU DON'T SAY WHAT TO DO WITH 20: no dice qué es lo que hay que hacer con 20).

Todos los procedimientos que hemos escrito hasta ahora han sido instrucciones. Para crear *operaciones* debemos hacer uso de la primitiva OUTPUT. Como ejemplo sencillo, he aquí un procedimiento que proporciona la distancia desde el origen de la tortuga; este procedimiento utiliza SQRT para devolver la raíz cuadrada de un número:

```
TO DISTANCIA
  OUTPUT SQRT (XCOR*XCOR+YCOR*YCOR)
END
```

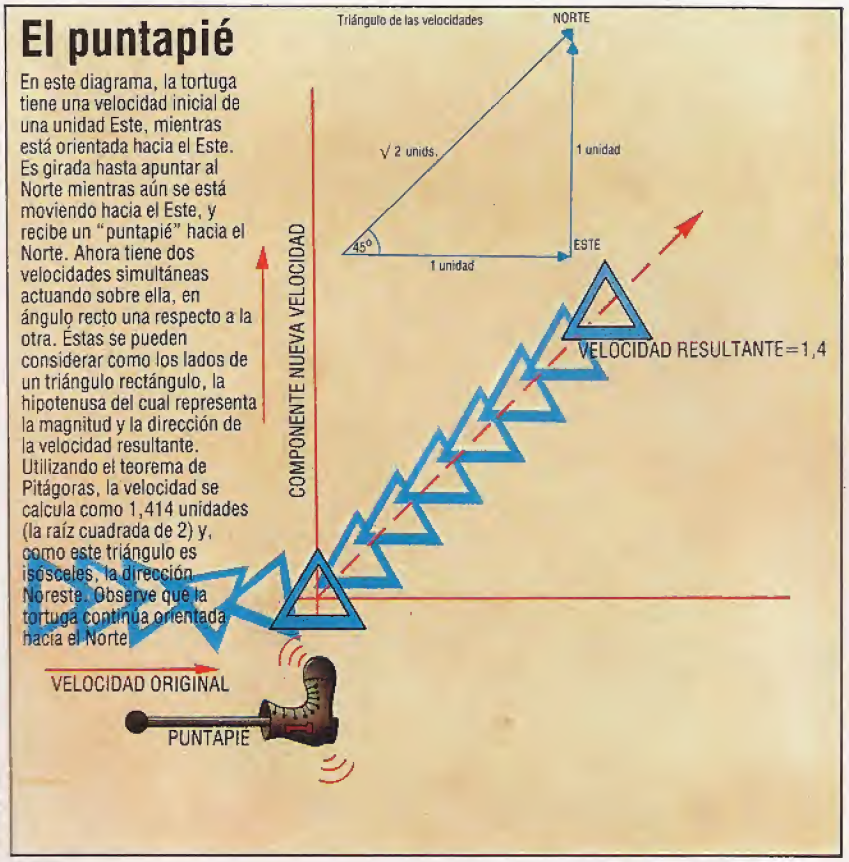
Intente desplazar la tortuga a distintas posiciones de la pantalla y utilice DISTANCIA para determinar a qué distancia se halla del origen. Por ejemplo, SETXY 30 40 PRINT DISTANCIA deberá dar como respuesta 50.

Cuando el LOGO ejecuta una instrucción OUTPUT, interrumpe la ejecución del procedimiento en curso, devolviendo el control al procedimiento que lo llamó. Esto se puede apreciar en el procedimiento MAX, que emite el mayor de dos números:

```
TO MAX :X :Y
  IF :X> :Y THEN OUTPUT :X
  OUTPUT :Y
END
```

El puntapié

En este diagrama, la tortuga tiene una velocidad inicial de una unidad Este, mientras está orientada hacia el Este. Es girada hasta apuntar al Norte mientras aún se está moviendo hacia el Este, y recibe un "puntapié" hacia el Norte. Ahora tiene dos velocidades simultáneas actuando sobre ella, en ángulo recto una respecto a la otra. Éstas se pueden considerar como los lados de un triángulo rectángulo, la hipotenusa del cual representa la magnitud y la dirección de la velocidad resultante. Utilizando el teorema de Pitágoras, la velocidad se calcula como 1,414 unidades (la raíz cuadrada de 2) y, como este triángulo es isósceles, la dirección Noreste. Observe que la tortuga continúa orientada hacia el Norte.



Kevin Jones

PRINT MAX 6 2 dará 6 como resultado. Intente escribir un procedimiento para obtener el valor absoluto de un número, de modo que tanto PRINT ABS 4 como PRINT ABS (−4) devuelvan el valor 4.

Nuestro juego le pedirá que digite su nombre y pulse Return. He aquí un procedimiento:

```
TO TOMAR.NOMBRE
  SPLITSCREEN
  PRINT1 [COMO SE LLAMA UD?]
  MAKE "NOMBRE FIRST REQUEST
  (PRINT "HOLA :NOMBRE)
END
```

REQUEST espera a que se entre una línea, acabándola con un Return. Luego da salida a la línea en forma de lista. FIRST proporciona el primer elemento de una lista. A continuación pruebe el procedimiento TOMAR.NOMBRE y digite "Luisa" como nombre. Luego vea qué sucede si utiliza "Luisa Moreno" como entrada.

El juego controla el movimiento en pantalla de la tortuga mediante las teclas R, L y K. R hará girar la tortuga 30° en el sentido de las agujas del reloj (derecha); L la hará girar la misma cantidad en el sentido contrario (izquierda); K se utiliza para "darle un puntapié" a la tortuga, es decir, para aumentar su velocidad sea cual sea la dirección en la que esté apuntando. La tortuga se moverá alrededor de la pantalla y necesitaremos que responda de inmediato a estas teclas. Sería una ayuda si hubiera alguna primitiva del LOGO (READKEY —leer tecla—, quizá) que proporcionara la última tecla que se hubiera pulsado. Si fuera éste el caso, podríamos escribir:

```
TO INSTRUCCION
  MAKE "INSTR READKEY
  IF :INSTR="R THEN RIGHT 30
  IF :INSTR="L THEN LEFT 30
  IF :INSTR="K THEN PUNTAPIE
END
```

Por desgracia, ¡esta primitiva no existe! Pero podemos escribirla como un procedimiento:

```
TO READKEY
  IF RC? THEN OUTPUT READCHARACTER
  OUTPUT"
END
```

Cuando se pulsa una tecla, ésta se almacena en el buffer del teclado. READCHARACTER simplemente saca el último carácter del buffer; si éste estuviera vacío, READCHARACTER esperaría a que se pulsara una tecla y después produciría el carácter correspondiente. RC? es cierto si el buffer contiene cualquier carácter, y es falso si está vacío. Así que READKEY producirá ahora el último carácter del buffer, o una palabra nula en caso de que éste esté vacío.

La tortuga espacial es en realidad una *tortuga dinámica*. Ésta es una tortuga que posee una velocidad, además de una posición y un encabezamiento como cualquier tortuga normal terrícola. La tortuga dinámica está en el espacio, de modo que no hay ni fricción ni gravedad. La tortuga dinámica obedecerá las leyes del movimiento de Newton.

Nuestra ilustración clarificará todo esto pero, a modo de ejemplo, vamos a suponer que la tortuga dinámica se está moviendo de izquierda a derecha a través de la pantalla a una velocidad de 1. Si se pulsa la tecla L, la tortuga gira para encararse hacia la parte superior de la pantalla, pero el impulso de la tortuga hará que se siga moviendo en su trayectoria horizontal. Entonces, si se pulsa K, la tortuga dinámica recibe un "puntapié" en la dirección en la cual está encarada. Esto resulta en un empuje hacia arriba de la pantalla de velocidad 1, y la tortuga dinámica se moverá diagonalmente a través de la pantalla a una velocidad de 1,4. La tortuga dinámica permitirá que usted experimente con un cuerpo que obedece las leyes de Newton; está diseñada para permitirle desarrollar una comprensión intuitiva de estas leyes sin que necesite comprender todas las ecuaciones correspondientes.

En el programa la velocidad de la tortuga dinámica se considera en términos de dos componentes a lo largo de los ejes *x* e *y*. Estos componentes se hallan utilizando las funciones SIN y COS. Los únicos controles del juego son los tres que ya hemos mencionado. Para empezar el juego, tan sólo digite START. Dispone de un tiempo fijo en el transcurso del cual debe alcanzar su objetivo, y el programa lleva el registro del mejor marcador conseguido.

Ian McKinnell

Tortuga extraterrestre
Tal como está impreso, el programa sólo generará la tortuga y el objetivo (su base). Las estrellas y los planetas que vemos en la fotografía se agregan utilizando algunos sencillos procedimientos de círculos

Complementos al LOGO

LOGO MIT	LOGO LCSI
DRAW	CS
PRINT1	TYPE
RC?	KEYP
READCHARACTER	RC
REQUEST	RL
SETHEADING	SETH
SETXY	SETPOS (seguido de lista)
FULLSCREEN	FS (inexistente en el Spectrum)
SPLITSCREEN	SS (inexistente en el Spectrum)

El LOGO LCSI, la sintaxis de IF es algo diferente; p. ej.:

IF DISTANCIA<5 [HAZ STOP]

Proyecto de programa

Escriba un programa para jugar al *Lunar lander* (véase p. 832). He aquí un breve resumen:
El jugador está pilotando un cohete que se halla a cierta distancia por encima de la superficie de la Luna y que lleva una cantidad limitada de combustible. La gravedad ejerce un empuje constante sobre su nave e incrementa su velocidad de descenso en función de una cantidad fija por segundo. Pulsando F se encienden los motores del cohete. El objetivo del juego consiste en utilizar la tecla F de modo que el descenso sea lo suficientemente lento como para realizar un alunizaje seguro.

Abreviaturas

OUTPUT	OP
PRINT	PR
READCHARACTER	RC
REQUEST	RQ
SETHEADING	SETH

Respuestas a los ejercicios

1. Triángulos anidados

```
TO TRI :TAMAÑO :NIVEL
  IF :NIVEL=0 THEN REPEAT 3[FD :TAMAÑO
     RT 120] STOP
  TRI(:TAMAÑO/2)(:NIVEL-1)
  FD(:TAMAÑO/2)
  TRI(:TAMAÑO/2)(:NIVEL-1)
  RT 60
  TRI(:TAMAÑO/2)(:NIVEL-1)
  FD(:TAMAÑO/2)
  RT 60
  TRI(:TAMAÑO/2)(:NIVEL-1)
  LT 60
  BK(:TAMAÑO/2)
  LT 60
  BK(:TAMAÑO/2)
END
```

2. Copo de nieve cuadrado

```
TO NIEVE 1 :TAMAÑO :NIVEL
  REPEAT 4[LADO1 :TAMAÑO :NIVEL RT 90]
END
TO LADO1 :TAMAÑO :NIVEL
  IF :NIVEL=0 THEN FD :TAMAÑO STOP
  LADO1(:TAMAÑO/3)(:NIVEL-1)
  LT 90
  LADO1(:TAMAÑO/3)(:NIVEL-1)
  RT 90
  LADO1(:TAMAÑO/3)(:NIVEL-1)
  RT 90
  LADO1(:TAMAÑO/3)(:NIVEL-1)
  LT 90
  LADO1(:TAMAÑO/3)(:NIVEL-1)
END
```

3. Curva sin gradiente en ningún punto

```
TO W :XPASO :YPASO :NIVEL
  WARRIBA :XPASO :YPASO :NIVEL
  WABAJO :XPASO :YPASO :NIVEL
END

TO WARRIBA :XPASO :YPASO :NIVEL
  IF :NIVEL=0 THEN SETXY (XCOR+:XPASO)
     (YCOR+:YPASO)STOP
  WARRIBA(:XPASO/6)(:YPASO/2)(:NIVEL-1)
  WABAJO(:XPASO/6)(:YPASO/2)(:NIVEL-1)
  WARRIBA(:XPASO/6)(:YPASO/2)(:NIVEL-1)
  WARRIBA(:XPASO/6)(:YPASO/2)(:NIVEL-1)
  WABAJO(:XPASO/6)(:YPASO/2)(:NIVEL-1)
  WARRIBA(:XPASO/6)(:YPASO/2)(:NIVEL-1)
END
TO WABAJO :XPASO :YPASO :NIVEL
  IF :NIVEL=0 THEN SETXY(XCOR+:XPASO)
     (YCOR-:YPASO)STOP
  WABAJO(:XPASO/6)(:Y PASO/2)(:NIVEL-1)
  WARRIBA(:XPASO/6)(:YPASO/2)(:NIVEL-1)
  WABAJO(:XPASO/6)(:YPASO/2)(:NIVEL-1)
  WABAJO(:XPASO/6)(:YPASO/2)(:NIVEL-1)
  WARRIBA(:XPASO/6)(:YPASO/2)(:NIVEL-1)
  WABAJO(:XPASO/6)(:YPASO/2)(:NIVEL-1)
END
```

La tortuga del espacio

```
TO COMENZAR
  MAKE"MAX 0
  MAKE"MEJOR"
  DRAW
  HT
  OBJETIVO
  JUGAR
END

TO OBJETIVO
  PU SETXY 0 5 PD
  RT 90
  REPEAT 36 [FD 31.4/36 RT 10]
  PU
END

TO JUGAR
  TOMAR.NOMBRE
  INIC
  CONDUCIR
END

TO TOMAR.NOMBRE
  SPLITSCREEN
  PRINT1 [COMO SE LLAMA UD?]
  MAKE"NOMBRE FIRST REQUEST
END

TO INIC
  MAKE"MARCADOR 200
  SETXY 100 100
  SETH 270
  MAKE"XVEL 0
  MAKE"YVEL 0
  FULLSCREEN
  ST
END

TO CONDUCIR
  ORDEN
  MOVIMIENTO.TORT.DIN
  IF DIST<5 THEN HAZ STOP
  MAKE"MARC:MARC-1
  IF :MARC=0 THEN T AGOTADO
  STOP
  CONDUCIR
END

TO ORDEN
  MAKE"ORD READKEY
  IF :ORD="R THEN RIGHT 30
  IF :ORD="L THEN LEFT 30
  IF :ORD="K THEN PUNTAPIE
END

TO READKEY
  IF RC? THEN OUTPUT
  READCHARACTER
  OUTPUT"
END

TO PUNTAPIE
  MAKE"XVEL+3*SIN HEADING
  MAKE"YVEL+3*COS HEADING
END

TO MOVIMIENTO.TORT.DIN
  SETXY XCOR+:XVEL
  YCOR+:YVEL
END

TO DISTANCIA
  OUTPUT SORT
  (XCOR*XCOR+YCOR*YCOR)
END

TO HAZ
  PRINT"
  SPLITSCREEN
  (PRINT[BIEN HECHO]:NOMBRE)
  (PRINT[SU MARCADOR FUE]
  :MARCADOR)
  INFORME
  OTRAVEZ
END

TO INFORME
  IF :MARC>:MAX THEN MAKE
  "MAX:MARC MAKE"
  MEJOR:NOMBRE PRINT"
  (PRINT[MARC MAS ALTO ES]
  :MEJOR[CON]:MAX[PUNTOS]
END

TO OTRAVEZ
  PRINT1[OTRA PARTIDA?]
  MAKE"RESP FIRST REQUEST
  IF :RESP="SI THEN REPETIR
     STOP
  IF :RESP="NO THEN STOP
  PRINT [DECIDETE, SI O NO?]
  OTRAVEZ
END

TO TIEMPO.AGOTADO
  PRINT"
  SPLITSCREEN
  PRINT[SU T HA CONCLUIDO]
  OTRAVEZ
END

TO REPETIR
  HT
  TOMAR.NOMBRE
  INIC
  CONDUCIR
END
```

Subir hasta cero

Dejando, por el momento, el análisis del direccionamiento, estudiaremos con más detalle cómo funcionan las pilas

Hasta ahora nuestro uso de los dos registros índices de pila, el S y el U, se ha limitado a servirnos de ellos como registros índices adicionales. Sólo muy de pasada hemos mencionado el empleo de la llamada *pila hardware* como almacén de direcciones de retorno tras las llamadas a subrutinas. Se impone volver sobre nuestros pasos y estudiar la arquitectura de una pila, así como el modo de utilizarla.

Una pila es un tipo específico de estructuración de datos dentro de la categoría de las *listas*. Aun cuando nada sepa de lenguajes cada vez más populares que se basan en el tratamiento de listas (p. ej., el LISP y el LOGO), usted tiene sin duda una idea común de lista, puesto que simplemente se trata de una secuencia de ítems de datos. Tal secuencia puede ordenarse según alguna propiedad común a los datos (así, p. ej., de menor a mayor si se trata de datos numéricos; por orden alfabético, si son formados con caracteres alfanuméricos), o bien puede ordenarse aleatoriamente según vayan integrándose a la lista. En todas las listas es claro que podemos distinguir entre "siguiente" y "anterior", y en particular entre el primer elemento de la lista y el último (llamados "cabeza" —*head*— y "cola" —*tail*—).

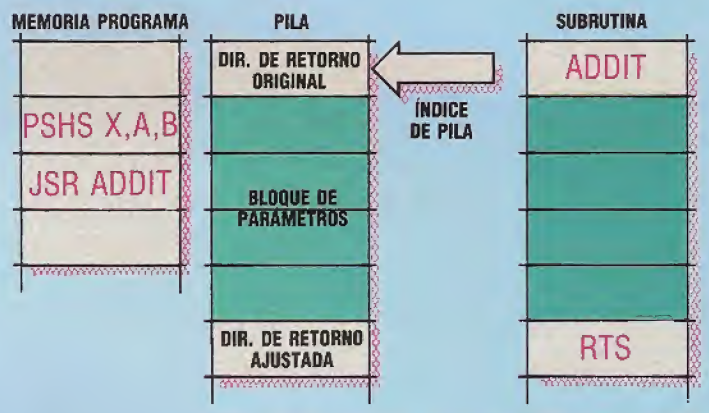
Una característica importante de una lista es que es una *estructuración dinámica de datos*; es decir, los ítems de datos pueden añadirse o eliminarse de ella arbitrariamente. En una lista general los datos se colocan o se sacan de cualquier posición que ocupen en ella. Lo que convierte una lista en *pila (stack)* es que los datos sólo pueden ser introducidos o extraídos por uno solo de sus extremos. Cada nuevo ítem añadido a la pila se convierte en "cabeza de lista", y sólo éste puede sacarse de ella.

Su mismo nombre nos da una idea del modo como funciona. Considérese una pila de platos en un cajón: según se van necesitando los platos éstos se van tomando por arriba, y encima sólo se ponen platos limpios. Desde luego que usted podría extraer un plato de la parte intermedia de la pila, pero esto sería absurdamente dificultoso. Lo que sí es posible es inspeccionar un determinado dato en cualquier posición de la pila.

En el funcionamiento de la pila pueden presentarse dos situaciones extremas: el caso en que se vacía la pila, donde no hay problema alguno si se añade un nuevo dato en la siguiente operación de pila, pero si ello no es así, puede resultar problemático; el otro caso extremo es que la pila se llene a rebosar. Esta segunda hipótesis es fácil de imaginar volviendo a nuestra pila de platos: habrá un momento en que ésta alcance el techo y sea imposible añadirle ningún plato más.

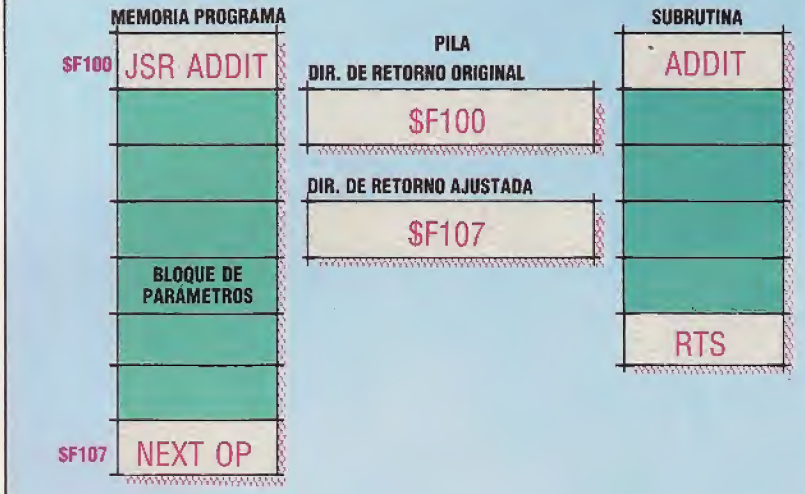
Las pilas de los ordenadores funcionan de modo análogo. Las operaciones de poner o de sacar datos

Poner parámetros



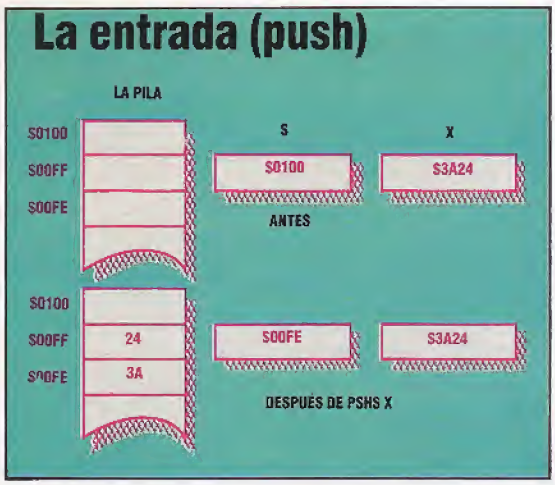
Los parámetros se pueden pasar a una subrutina si los cargamos en registros y los colocamos en la pila. La subrutina puede sacarlos de allí si cuidamos que la dirección de retorno de JSR sea desplazada convenientemente hacia abajo después de haberse accedido a los parámetros. Si no se hace así, la pila crecerá cada vez más hasta que se produzca un desbordamiento

Inserción de parámetros

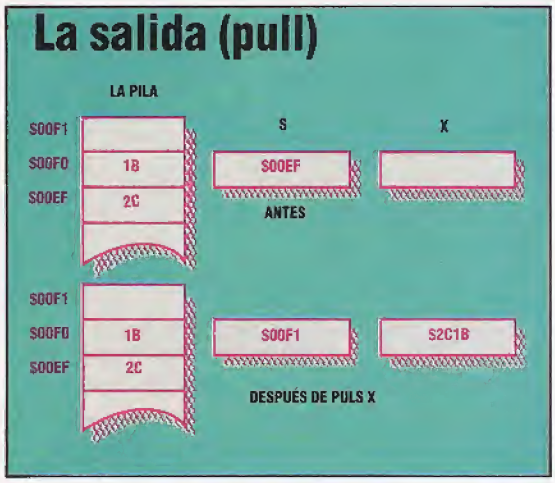


Un método más práctico de pasar parámetros consiste en insertarlos en el programa directamente, después de la llamada JSR a la subrutina. En este caso la subrutina puede emplear la dirección de retorno que está en la pila como dirección base del bloque de parámetros y acceder a ellos por medio del direccionamiento indexado. La dirección de retorno debe ser ajustada para que pueda indicar la instrucción siguiente en el programa y no el inicio del bloque de los parámetros

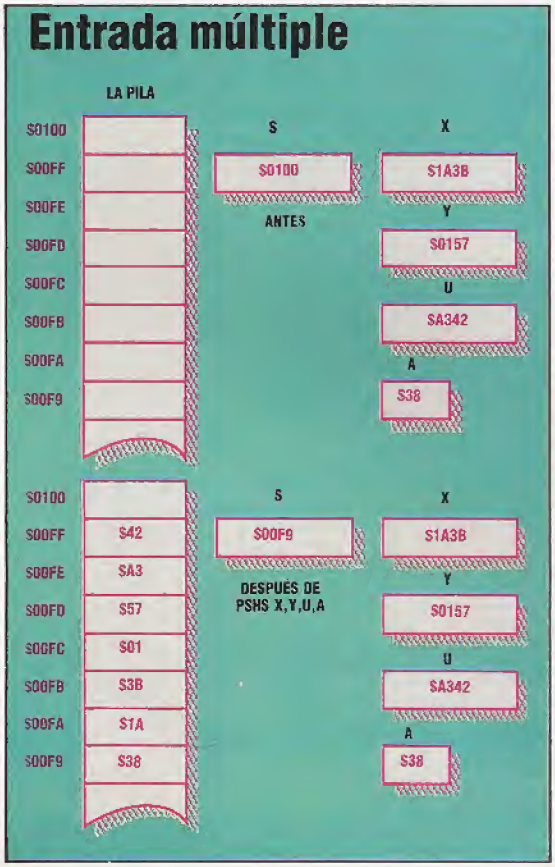
Empujón va
El índice de pila del 6809 siempre apunta a la "cima" (*top*) de la pila, es decir, al último byte que entró. Una vez ejecutada PSHS X, entonces S se decrementa en dos unidades, para poder indicar la nueva cima, mientras que el contenido de X (que es un registro de dos bytes) es escrito en dicha dirección según el formato *hi-lo*. Luego, efectivamente, la pila "sube hasta cero", pues el indicador de pila señala las posiciones inferiores de la memoria según va creciendo la pila



Empujón viene
Una vez ejecutado PULS X, el contenido de los dos bytes que están en la dirección del índice de pila es copiado en X, mientras que S es incrementada en dos unidades para indicar la nueva cima de la pila



En tropel
Cuando se ejecuta una operación de pila con múltiples registros, los registros en cuestión son accedidos según un orden predeterminado: PC,U o bien S,Y,X,DP,B,A,CC. Si es ejecutada PSHS X,Y,U,A, los primeros en entrar en la pila serán los contenidos de U seguidos de Y,X y A



Liz Dixon

de ellas se conocen como operación de "poner" y de "sacar" (*pushing* y *pulling/popping*) respectivamente. Y las situaciones extremas mencionadas se conocen como "desbordamiento" (negativo o normal) de datos (*underflow* y *overflow*).

La implementación de las pilas puede realizarse de varias maneras (p. ej., empleando una matriz de una dimensión en un programa BASIC), pero el método que estamos estudiando exige un bloque de memoria disponible y un registro que podemos designar como *índice de pila*. Este apuntador se hace imprescindible para conocer la posición que ocupa el dato en cabeza. A diferencia de la pila de platos, la de una memoria no puede ser inspeccionada porque no existe ningún indicio que permita distinguir una posición que contenga un determinado dato de la siguiente a ella, la cual hasta puede que no forme parte de la pila. Vale la pena señalar que, al igual que en una operación de "carga" de un registro desde memoria, donde en realidad el dato no es "extraído" de la memoria, sino copiado, así también los datos en realidad no son "sacados" de la pila, sino que se cambia el índice de la parte superior de ésta.

Por tanto, el índice de pila contiene la dirección de la actual cima (*top*) de la pila. Dos son las posibles variaciones: éste puede facilitar la dirección de la siguiente posición libre para almacenar datos o bien la dirección del último ítem de datos almacenados en la pila. Esta última indicación es la que se ha escogido por convenio para el 6809, sabiendo que no existen ventajas al preferir éste en lugar del otro método: otros procesadores emplean la técnica alternativa.

Entre una pila de platos y la de memoria existe una diferencia significativa en cuanto a organización, y es que en esta última y dentro del sistema del 6809 el crecimiento es hacia abajo: cuantos más datos van poniéndose a la pila, más bajas son las direcciones señaladas por el índice. Es lo que se entiende por "crecer hasta cero".

Operaciones de la pila

Las dos operaciones de una pila del 6809 se representan por PSH (*push*: poner) y PUL (*pull*: sacar). Ambas pueden emplearse con cualquiera de los dos índices, el S y el U, por lo que distinguiremos entre PSHS, PULS, PSHU y PULU. Los datos sobre los que se opera deben proceder de (o ir a) un registro, aunque es posible poner o sacar de varios registros a la vez mediante una sola instrucción.

La instrucción PSHS X primero hará que S se decremente, o sea el índice de pila se decrementa en dos unidades (o en una, si el registro colocado en la pila es de ocho bits) para apuntar a la dirección de la siguiente posición libre de pila; y además hará que el contenido de X quede almacenado en dicha dirección. El primer diagrama ilustra esta operación. De nuevo es de notar el convenio del direccionamiento *hi-lo* para el 6809: el byte superior, o *hi*, de X (o sea, $3A) queda almacenado en $00FE, es decir, una posición más abajo que el byte inferior, o *lo*, que es $24 y queda almacenado en $00FF. Si se sirve de un ensamblador, estos detalles de si los indicadores de pila incrementan o decrementan carecen de interés: el ensamblador se encarga de preparar convenientemente la memoria.

La instrucción PULS X tiene el efecto contrario: el

valor de 16 bits contenido en ese momento en S es cargado en X, incrementando seguidamente S en dos unidades. Es lo que ilustra el segundo diagrama.

Se puede poner (o sacar) más de un registro a la vez. Por ejemplo en la siguiente instrucción:

```
PSHS X,Y,U,A
```

En casos como el presente en que más de un registro es colocado en la pila, se desprecia el orden en que son especificados los registros y en su lugar se observa este otro orden: PC (registro contador del programa), U o bien S,Y,X,DP (registro de página directa), B,A y CC (registro de código de condición). Naturalmente son sacados por este mismo orden pero a la inversa. La única limitación consiste en que ni S ni U pueden ser colocados en la pila.

Las pilas se emplean en la programación general por ser lugares muy convenientes para el almacenaje rápido temporal, pero el principal servicio que prestan se revela al tratar con interrupciones (de las que hablaremos más adelante) y subrutinas. Ya hemos visto cómo es colocado en la pila el contenido del registro contador del programa en el momento de llamar una subrutina, y cómo se extraía de ella cuando se abandonaba ésta (RTS equivale a PULS PC). Cualquiera de las dos pilas puede servir para pasar parámetros a la subrutina, pero en especial S.

Al método hasta ahora seguido para pasar parámetros a través de los registros (como, p. ej., el programa de la Tabla de Salto de p. 1119) suelen hacerse dos reparos importantes. El primero es que puede haber más parámetros que registros, y el segundo es que puede resultar engorroso el hecho de que la rutina llamada emplee un registro donde se encuentre un parámetro que es necesario conservar. Hay, sin embargo, otras dos técnicas conocidas para pasar parámetros:

1) Los datos pueden almacenarse junto al código del programa, justo después de la llamada a la rutina, mediante el uso de las directivas FCB, FDB o FCC. El valor del registro contador del programa, que es llevado a la pila por medio de la instrucción JSR, facilita la dirección del primero de estos valores (ya que PC siempre señala el byte siguiente al de la instrucción en curso) y con desplazamientos apropiados sirve para obtener las restantes direcciones. El primer programa que damos como ejemplo ilustra esta técnica. Habrá de cuidarse que se organice de tal modo la instrucción RTS que pase el control a una instrucción real y no a un grupo de datos.

2) Los datos pueden ser cargados en registros y ser llevados a la pila antes de la llamada de la subrutina, y de ella se sacarán para ser utilizados en la subrutina. Aquí la atención se pondrá en que con la instrucción RTS el índice de pila accederá a la dirección de retorno del PC previamente guardado en la pila. Esta operación se ilustra en el segundo ejemplo. Se trata de un método mucho más útil que el primero.

En ambos procedimientos, el doble papel desempeñado por S y U, como registros índices y como índices de pila, significa que los datos llevados a la pila pueden ser referenciados por medio del direccionamiento indexado además de poder accederse fácilmente a ellos para ser sacados de la pila. Lo cual permite asegurar que se dejaron en la pila los ítems correctos para el momento del regreso.

Suma de precisión múltiple

He aquí dos fragmentos de lenguaje máquina para ilustrar dos métodos alternativos de realizar una suma de precisión múltiple por medio de pilas. En el primer fragmento, los parámetros son colocados en ellas después de ser llamada la subrutina. Una llamada para sumar dos números de cuatro bytes en $100 y $104, dejando el resultado en $108, sería:

```
        BSR     MPADD
        FCB     4       ; Long. de cada núm. (bytes)
        FDB     $100    ; Dir. del primer número
        FDB     $104    ; Dir. del segundo número
        FDB     $108    ; Dir. del resultado
        ORG     $1000
MPADD   PSHS    X,Y,U,A,B,CC ; Guarda todos los registros (introduce 9 bytes en la pila)
        LDU     9,S     ; Cargará en U la dir. de retorno de PC colocada en la pila por la instrucción BSR
        PULU    X,Y,B   ; Se tratan datos que siguen a la llamada de subrutina como si fueran una pila, de modo que se obtienen las dirs. de los dos núms. y se introducen en X e Y, así como la long. en B
        LDU     ,U      ; Dir. del resultado queda en U
        ANDCC   #%11111110 ; Limpia el flag de arrastre
BUCLE   LDA     ,X+     ; Realiza suma de un byte más otro, con posibilidad arrastre
        ADCA    ,Y+
        STA     ,U+     ; Almacena el resultado
        DECB
        BNE     BUCL    ; Compr. si suma ha concluido
        LDU     9,S     ; Si no lo está, BUCLE
        LEAU    7,U     ; De otra manera toma dir. de retorno y añade 7 para pasar por encima de los 7 bytes de parám. almac. tras la llamada
        STU     9,S
        PULS    PC,U,X,Y,A,B,CC ; Restaura todos los registros al estado que tenían antes, incluido PC en vez de usar un RTS de más
```

Este segundo ejemplo realiza la misma operación pero colocando los parámetros en la pila. La secuencia de llamada sería:

```
        LDU     #$108   ; Dirección del resultado
        LDX     #$100   ; Dirección del operando
        LDY     #$104
        LDA     #4      ; Longitud
        PSHS    U,X,Y,A ; Coloca los parámetros en la pila
        BSR     MPADD
        ORG     $1000
MPADD   PSHS    X,Y,U,A,B,CC ; Guarda todos los registros
        LEAU    11,U    ; Guardados los registros, toma 9 bytes, y la dirección de retorno toma 2 bytes, en total 11 bytes. Así la actual U apunta a los parámetros
        PULU    X,Y,B
        ANDCC   #%11111110
BUCLE   LDA     ,X+     ; Procede como anteriormente
        ADCA    ,Y+
        STA     ,U+
        DECB
        BNE     BUCLE
        PULS    PC,U,X,Y,A,B,CC
```

La guerra nuclear ha comenzado

Este terrorífico juego transcurre durante una conflagración nuclear y consiste en impedir la destrucción de la humanidad

El éxito de un juego diseñado para salas recreativas a menudo depende de lo acertada que sea su modalidad de "atracción" (la visualización que aparece en la pantalla mientras no se está jugando). El juego debe, asimismo, producir una adicción instantánea. Si fracasa en uno solo de estos aspectos será desechado enseguida por el propietario de la sala en favor de una máquina más rentable. A pesar de que ya ha pasado el momento en que *Comando de misiles (Missile command)* alcanzara su máximo éxito en las salas recreativas, durante un tiempo fue muy popular. La versión para las máquinas Atari es testimonio de su antigua gloria.

El escenario del juego es sencillo y sutil a la vez. El jugador se sitúa en la posición del comandante de una estación antimisiles durante una guerra nuclear y debe proteger a seis ciudades de la amenaza de destrucción haciendo explotar misiles antinucleares que salen al paso de las cabezas explosivas que se acercan. El diseño pretende satisfacer tanto la megalomanía como el ansia de aventuras latentes en un aficionado a los juegos de "marcianitos", se encuentre éste en una sala recreativa o en su casa.

Durante el juego, la pantalla muestra una vista de sección transversal de la acción, que ilustra las seis ciudades y una estructura piramidal en el centro con los misiles antibalísticos listos para su lanzamiento. El rastro de los misiles que se acercan aparece luego desde la parte superior de la pantalla. El jugador desplaza una cruz alrededor de ésta valiéndose de la palanca de mando. La cruz se coloca en el recorrido de los misiles que se aproximan y, accionando el pulsador de disparo, se lanza un misil antibalístico desde la base de lanzamiento del jugador. El proyectil estalla en las coordenadas de la cruz, destruyendo todos los misiles que se acerquen y estén dentro del radio de alcance de la explosión.

Sin embargo, algunos misiles enemigos están provistos de cabezas explosivas múltiples que se dividen en varias trayectorias, cada una de las cuales puede destruir una ciudad entera. El juego se complica aún más con la aparición de satélites "asesinos" y bombarderos enemigos que vuelan a poca altura, capaces de lanzar nuevas olas de proyectiles nucleares. Se otorgan puntos por la destrucción de aviones y misiles enemigos. Al final de un ataque, un marcador muestra la cantidad de ciudades que uno ha conseguido proteger con éxito de la aniquilación y cuántos misiles antibalísticos le quedan.

A medida que el jugador va avanzando a través de los seis niveles de dificultad que posee el juego, los misiles atacantes se desplazan con mayor rapidez y va aumentando la cantidad de cabezas explosivas en las que se dividen. En este punto es necesario desarrollar una estrategia general, en vez de destruir cada misil por separado. El jugador puede optar, por ejemplo, por disponer una "barrera" de misiles antibalísticos que exploten en línea y, con suerte, neutralicen la oleada en una sola acción contraofensiva.

Además, en los niveles superiores el juego se complica por la aparición de cabezas explosivas lanzadas con paracaídas. Éstas son sumamente difíciles de destruir: si su misil estalla ligeramente fuera del objetivo, es probable que sea "soplado" fuera de ruta (presumiblemente debido a la corriente de aire ascendente) y, por lo tanto, es necesario hacer explotar un misil justo encima de uno de éstos.

En el transcurso del juego debe recordar que sólo dispone de un número limitado de misiles antibalísticos (30 en el primer nivel) y si éstos se acabaran estará condenado a presenciar cómo el enemigo aniquila la base de misiles y las ciudades.

Cada nivel se compone de dos olas de ataque separadas, después de las cuales se calcula el marcador. Al igual que en otros juegos Atari, se puede "saltar" a un nivel de juego superior. Cada nivel se distingue por sus diferentes colores de fondo y primer plano. El juego termina cuando se destruyen las seis ciudades, y esta conclusión inequívocamente pesimista se refuerza con la pantalla final, que visualiza una apocalíptica explosión y las palabras *THE END*.

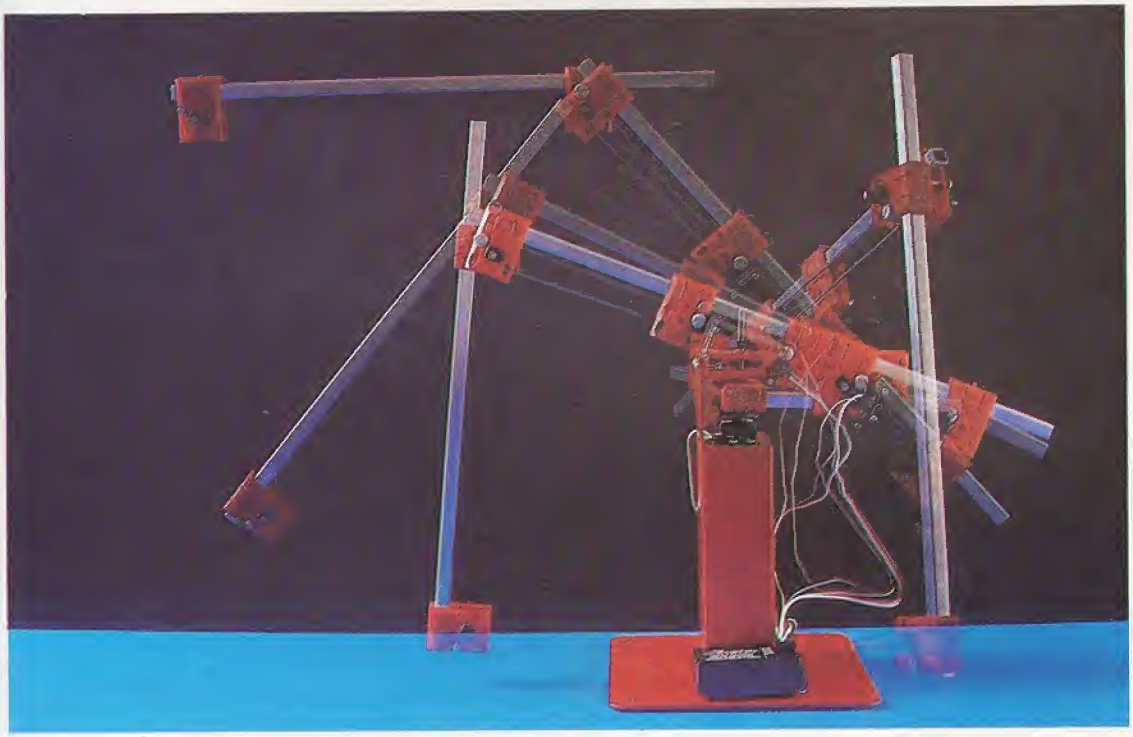
El juego se vende en cartucho para los ordenadores Atari y viene con un manual de instrucciones en color claramente superior a la documentación que acompaña a la mayoría de los otros juegos.

Comando de misiles: Para los ordenadores Atari
Editado y distribuido por: Atari, AUDELEC, Compás de la Victoria, 3, 29012-Málaga, España
Autores: Atari
Palanca de mando: Necesaria
Formato: Cartucho

Liz Dixon

Brazos y manos

Ahora consideraremos un aspecto básico del diseño de un robot: el control y movimiento de las extremidades superiores



Ian McKinnell

Juntura doble
Durante la próxima década, el robot arquetípico será el brazo simple equipado con diversas "manos" para uso industrial, doméstico y para aficionados. Son muy pocas las aplicaciones que realmente requieren la máquina pensante autónoma y autopropulsada tan corriente en la ciencia-ficción, pero una uña semiinteligente programable es un dispositivo tan significativo como lo fueran en su tiempo el arado o el telescopio

La eficacia de un robot depende en gran medida de la precisión con la cual pueda manipular objetos. Muchos robots se utilizan fundamentalmente para operaciones de "asir y colocar": trasladar componentes, en una planta industrial, de una cinta transportadora a otra, por ejemplo. Por tanto, el diseño del brazo-robot reviste la máxima importancia.

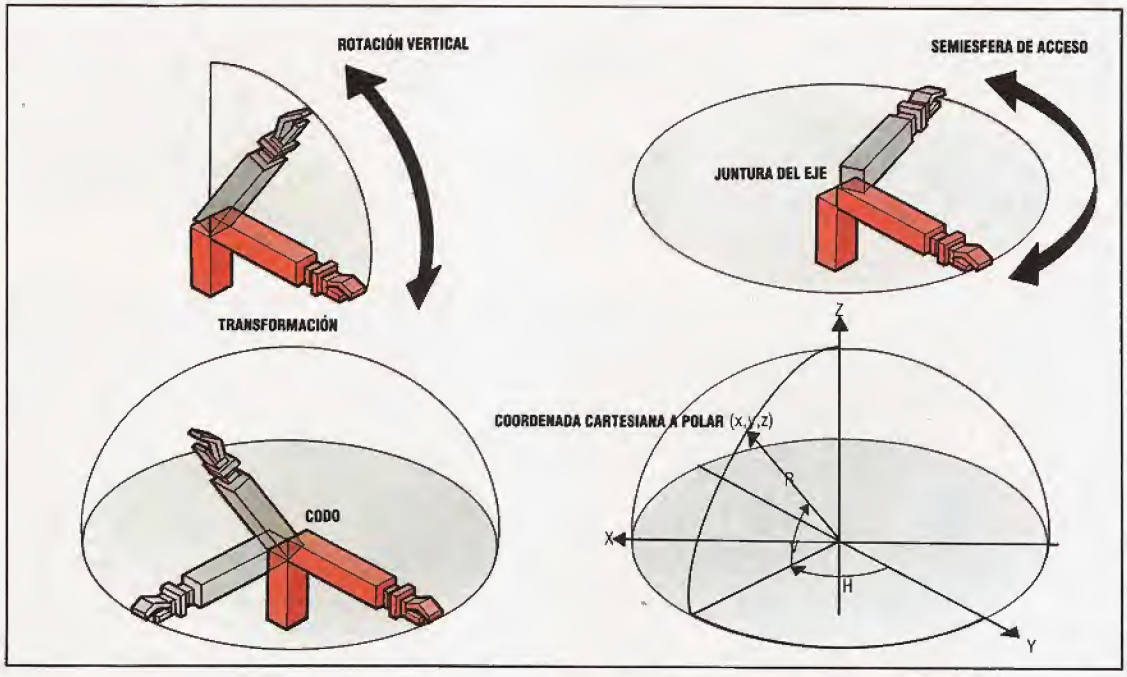
En general, son tres las exigencias que se deben considerar. Se debe desarrollar un sistema que describa la posición del brazo en cualquier momento dado; éste debe tener un "esqueleto" y debe haber un sistema "muscular" que accione el brazo y permita controlarlo. Las diferentes formas en las que interactúan estos tres elementos básicos suelen conformar la apariencia global de los brazos-robot. Sin embargo, en líneas generales se podría hacer una clasificación de distintos tipos de brazo atendiendo a los procedimientos espaciales utilizados para describir la posición del brazo en un momento dado.

En nuestro análisis del movimiento del robot (véase p. 1101), describimos el sistema de coordenadas cartesianas. Empleando este método, la posición del robot sobre el suelo se especificaba en virtud de dos ejes (x e y) en ángulo recto el uno del otro. A un brazo-robot se puede aplicar el mismo principio; pero, puesto que un brazo se puede mover libremente en tres dimensiones, es necesario que agreguemos otra variable (z) para describir la posición vertical del mismo. Utilizando estas coordenadas x, y y z podemos describir la posición del brazo en cualquier punto del espacio (siendo "espacio" simplemente la forma matemática de describir una superficie abierta).

Se puede construir un brazo-robot que se mueva exactamente a lo largo de estas tres coordenadas: lo que resulte se parecerá a una grúa de caballete elevada que se puede mover arriba/abajo, delante/atrás y de lado a lado (o en una combinación de las tres direcciones). Brazos como éste son muy apropiados para tareas en las que el trabajo se realiza en un área fija. Por ejemplo, al robot se lo puede colocar frente a una mesa de trabajo en la cual lleve a cabo todas sus tareas y, en este caso, un brazo cartesiano es más que adecuado. Pero este procedimiento tiene sus desventajas. Por ejemplo, los brazos de esta clase requieren una estructura especial del entorno, que los incapacita para realizar aplicaciones alejadas de la mesa de trabajo.

Otro método para describir la posición de un brazo emplea coordenadas cilíndricas. Para formarse una idea de cómo funcionan éstas, imagínese un bote de hojalata vacío; es fácil comprender que cualquier posición dentro del bote se puede describir especificando su distancia desde el centro del bote (utilizando una variable distancia, r); a qué distancia alrededor del bote está respecto a un punto fijo (empleando una variable angular, θ); y a qué distancia del lado del bote está (utilizando otra variable de distancia, z). De modo que recurriendo a las coordenadas cilíndricas sería fácil desarrollar un sistema que permitiera asir un objeto situado en una posición especificada dentro del bote.

Rotaciones robóticas
El brazo más simple, compuesto por una uña y una juntura de codo de dos ejes, es capaz de un posicionamiento de precisión en un volumen de espacio muy grande, tal como reflejan estas ilustraciones.
El codo gira sobre un gozne, permitiendo el movimiento vertical semicircular, y gira sobre un eje, lo que permite el movimiento circular horizontal. El brazo se desplaza hasta cualquier punto de la semiesfera de acceso mediante rotaciones en el eje y el gozne. Éstas se pueden obtener por trigonometría desde las coordenadas cartesianas (*x*, *y* y *z*) del punto tal como se indica: H, la rotación del eje, es igual a ARCTAN(*x*/*y*), mientras que V, el ángulo del gozne, es ARCSIN(*z*/R). El brazo se programa con las posiciones cartesianas de los objetos; transforma cada grupo de coordenadas en dos rotaciones que envía a sus servomotores, produciendo el movimiento



Kevin Jones

Los brazos que utilizan coordenadas esféricas llevan este proceso un paso más allá al especificar una posición en términos de dos ángulos y una distancia. En este caso, "distancia" equivale a la longitud del brazo, y los dos ángulos son la cantidad en función de la cual rota la base y el ángulo de elevación del brazo. Los brazos de este tipo se parecen mucho a una torreta de cañón, en la que se pueda variar la longitud del tubo del mismo. Las coordenadas esféricas se describen, por lo general, como *r*, θ y φ.

Para el ingeniero en robots es bastante sencillo diseñar un brazo que se pueda mover hacia dentro y hacia fuera telescópicamente, accionado, por ejemplo, mediante energía hidráulica.

El último y más corriente procedimiento para describir la posición de un brazo recurre a la utilización de otro tipo específico de coordenadas. Éste es un sistema diseñado especialmente para controlar brazos-robot imitando el accionar del brazo humano. Como antes, se necesitan tres variables para especificar la posición del brazo; esta vez son todas ángulos y se podrían describir como coordenadas θ, φ y γ. θ (*theta*) corresponde al ángulo a través del cual rota la base; φ (*phi*) representa el ángulo de elevación del brazo; y γ (*gamma*) describe el ángulo de una segunda juntura del brazo.

El sistema de coordenadas elegido determinará el tipo de "esqueleto" que requiera un robot. Todo lo que se necesita ahora es un poco de "músculo" para darle energía al movimiento del brazo. En general, son tres los tipos de músculos que se utilizan para los robots: eléctricos, hidráulicos y neumáticos. Analicémoslos separadamente.

Ya hemos hablado de la energía eléctrica en relación al movimiento del robot. Los mismos motores eléctricos paso a paso se podrían emplear para los brazos-robot. Por ejemplo, se pueden utilizar directamente, teniendo un potente motor en cada juntura del brazo y dejando que éste rote una pequeña cantidad para cada movimiento de juntura, o indirectamente, mediante engranajes, poleas o palancas.

Sin embargo, un sistema mejor implicaría hacer que los "músculos" del robot trabajaran de forma muy similar a los nuestros: estirándose y contrayéndose de modo que actúen directamente sobre el esqueleto del brazo. Esto se consigue disponiendo una serie de pistones para actuar sobre cada juntura del brazo. Estos pistones pueden ser hidráulicos (utilizando líquido) o neumáticos (empleando aire). Para los robots industriales se prefiere la energía hidráulica, porque puede proporcionar una presión mucho mayor (confiriéndole más fuerza al brazo) y porque el líquido no se dilata ni se contrae en la misma medida que el aire.

Esto significa que cuando un pistón se mueve a lo largo de un cilindro mediante presión hidráulica, no sufre efectos de "rebote", sino que se detiene exactamente en el punto deseado. El aire, por el contrario, no permite un posicionamiento de tanta precisión. Independientemente del sistema que se aplique, se pueden utilizar pistones de actuación simple o doble para producir movimiento en el brazo. Este tipo de fuerza motriz se denomina *de acción lineal*.

Aún es posible otro refinamiento. En vez de utilizar pistones que se muevan hacia atrás y hacia delante y luego traducir este movimiento en una rotación de la juntura, se puede recurrir a un *actuador rotatorio*. Éste produce una rotación directa en las junturas por medio de presión sobre un aspa dentro de una envoltura circular. Se trata de un proceso similar al empleo de un motor eléctrico paso a paso, pero la presión hidráulica significa que se puede ejercer muchísima más fuerza. La presión neumática no es apropiada para estas aplicaciones.

Una vez que se ha tomado una decisión sobre la mecánica del brazo-robot, ya sólo hace falta una "mano" (o *efector final*), de modo que, una vez posicionado correctamente el brazo, éste pueda hacer algo. Aquí resulta de ayuda pensar en la forma en que trabaja la mano humana. Consideremos la muñeca: si estuviera recubierta de yeso de manera que no se pudiera mover, la mayoría de las tareas resultarían mucho más difíciles. Cuando se pulsa un teclado, por ejemplo, las muñecas permiten que las manos se muevan hacia arriba y hacia abajo mien-

tras se golpean las teclas; esto se conoce como *cabezada* y sin ella al mecanografiar se tendría que mover todo el antebrazo hacia arriba y hacia abajo.

Las muñecas también se mueven hacia uno y otro lado a medida que se van pulsando las distintas teclas; esto se denomina *guiñada*, y su ausencia supondría el movimiento del codo. Cuando se termina de teclear, se pueden girar las muñecas de modo que las manos descansen con el pulgar hacia arriba junto al teclado. Esto se conoce como *tonel* y, de no disponer del movimiento de muñeca, sería necesario un complicado juego de movimientos de hombro.

En un plano ideal, estos tres distintos movimientos se deberían incorporar en la muñeca del robot. Cada uno de los movimientos (cabezada, guiñada y tonel) puede actuar en dos direcciones (arriba/abajo, izquierda/derecha, en sentido horario/antihorario) y cada combinación de movimiento y dirección se conoce como *grado de libertad*. En consecuencia, se puede decir que un robot que incorpore cabezada, guiñada y tonel posee seis grados de libertad. Los robots se construyen con menos grados de libertad, generalmente cuatro o cinco, pero cada reducción en el movimiento de muñeca queda compensada con un aumento de los movimientos que deben efectuar los otros miembros, más largos, del brazo.

La mano del robot

Ahora debemos considerar el diseño de la mano propiamente dicha. La configuración ideal sería una mano similar a la humana dispuesta en el extremo de un brazo semejante al del hombre; ya es posible hallar manos-robot que respondan a esta definición. La forma más común de mano-robot es una uña de tres dedos (compuesta por dos dedos más el "pulgar" opuesto) que permite que el robot coja objetos de una forma muy parecida a como lo haría una mano humana.

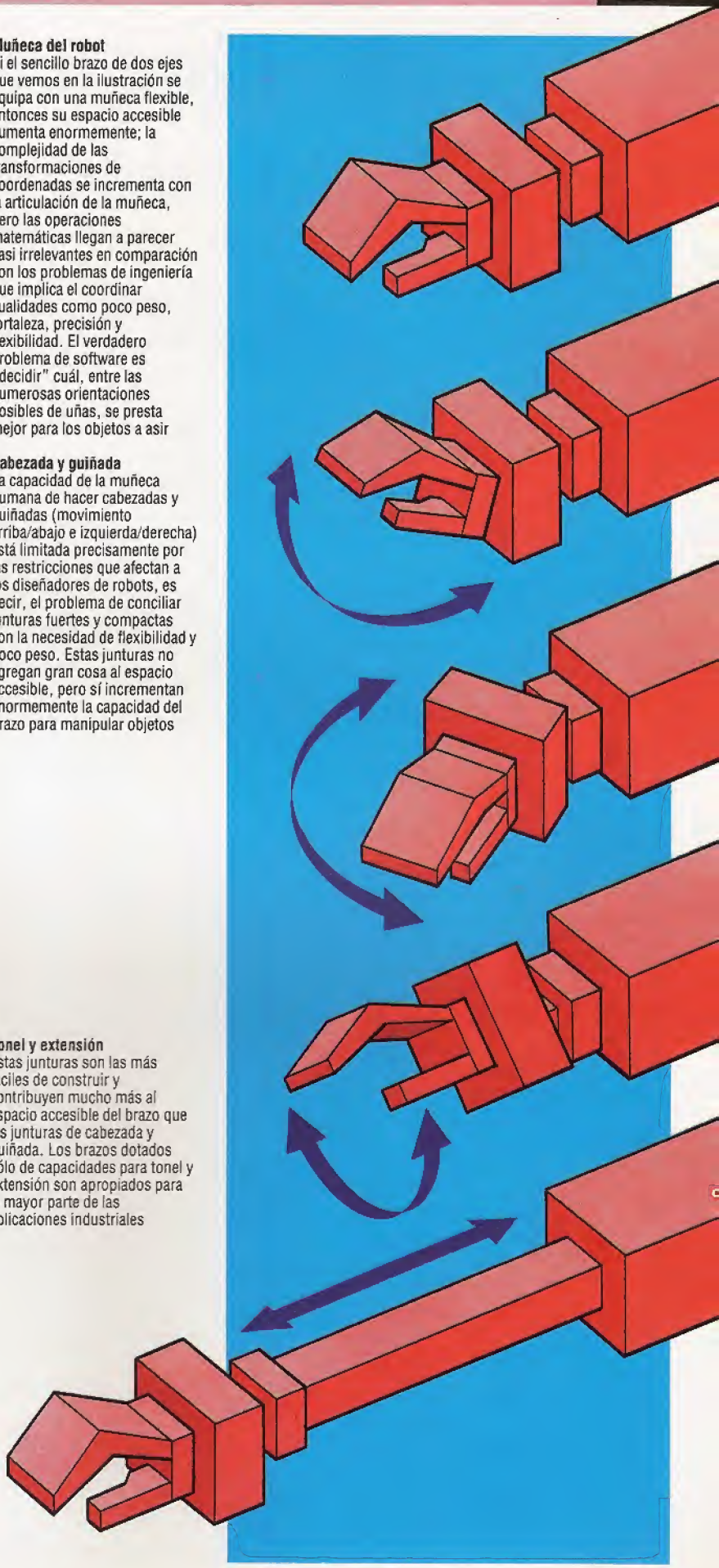
La energía que se utiliza para activar la mano puede ser de cualquiera de los tres tipos ya mencionados, y dependerá de la tarea que haya de llevar a cabo el robot. Si la mano debe mover objetos grandes que pesen alrededor de un centenar de hilos, probablemente será necesaria la hidráulica. Pero para muchas aplicaciones bastará la energía eléctrica o neumática, porque la mano sólo necesitará asir el objeto y soltarlo cuando se desee: si el brazo y la muñeca han posicionado correctamente la mano, ello no requerirá ninguna precisión; un simple movimiento de apertura y cierre será suficiente.

En muchos casos, no obstante, el brazo-robot no estará dotado de mano. Ya hemos empleado los términos "efector final" para describir una mano, pero estas palabras pueden igualmente aludir a otras muchas cosas. Un robot que se emplee para soldar no requiere mano en absoluto: se puede instalar una pistola soldadora directamente en la muñeca. En realidad, algunos robots son capaces de elegir el efector final correcto para la tarea que estén llevando a cabo; pueden descartar un efector final (un destornillador, supongamos) e insertar otro (una pistola spray, p. ej.) en un conector estándar en la muñeca. Puede que ésta no sea una acción de índole particularmente humana, pero sirve para hacer que los robots sean en extremo adaptables.

Muñeca del robot
Si el sencillo brazo de dos ejes que vemos en la ilustración se equipa con una muñeca flexible, entonces su espacio accesible aumenta enormemente; la complejidad de las transformaciones de coordenadas se incrementa con la articulación de la muñeca, pero las operaciones matemáticas llegan a parecer casi irrelevantes en comparación con los problemas de ingeniería que implica el coordinar cualidades como poco peso, fortaleza, precisión y flexibilidad. El verdadero problema de software es "decidir" cuál, entre las numerosas orientaciones posibles de uñas, se presta mejor para los objetos a asir

Cabezada y guiñada
La capacidad de la muñeca humana de hacer cabezadas y guiñadas (movimiento arriba/abajo e izquierda/derecha) está limitada precisamente por las restricciones que afectan a los diseñadores de robots, es decir, el problema de conciliar junturas fuertes y compactas con la necesidad de flexibilidad y poco peso. Estas junturas no agregan gran cosa al espacio accesible, pero sí incrementan enormemente la capacidad del brazo para manipular objetos

Tonel y extensión
Estas junturas son las más fáciles de construir y contribuyen mucho más al espacio accesible del brazo que las junturas de cabezada y guiñada. Los brazos dotados sólo de capacidades para tonel y extensión son apropiados para la mayor parte de las aplicaciones industriales



Sprites en LOGO

Después de haber estudiado con cierto detalle la geometría de tortuga, avanzaremos en nuestro curso examinando el empleo de sprites

Utilizando LOGO, los sprites se comportan de forma similar a la tortuga, obedeciendo todas las instrucciones que cumple ésta. Sin embargo, a diferencia de la misma, nosotros podemos definir la forma de un sprite, si bien estas formas no rotan en la pantalla cuando cambia el encabezamiento del sprite como lo suele hacer una tortuga.

En LOGO Commodore, la tortuga se cuenta como el sprite número 0, y hay otros siete sprites (numerados del 1 al 7). Para empezar, el sprite 0 es el sprite "en curso" y obedece todas las instrucciones para sprites que se generan. Para hacer que el sprite 1 sea el sprite en curso, simplemente debe ejecutarse TELL 1. A partir de entonces, el sprite 1 obedecerá todas las instrucciones para sprites hasta que se especifique un sprite en curso diferente.

No obstante, después de ejecutar TELL 1, no se verá nada en la pantalla. Ello se debe a que todos los sprites, excepto la tortuga, empiezan como objetos "ocultos" y tienen sus "lápices" hacia arriba. Para poder ver el sprite 1 y su trayectoria, debe ejecutarse ST, y en la pantalla aparecerá un cuadrado impreciso. Experimente con esta cuadrícula de sprite utilizando las instrucciones de tortuga FD,BK,RT,LT,PD,PU,ST,HT, etc.

Si se desplaza el sprite 1 hasta la misma posición que el sprite 0 (la tortuga), notará que el sprite 1 parecerá que está detrás de la tortuga. En general, los sprites de número inferior se muestran "por delante" de los sprites de números superiores. Esto es muy útil para crear efectos tridimensionales.

En el disco de utilidades del LOGO Commodore hay un editor de sprites. Cárguelo digitando READ"SPRED. Para editar la forma del sprite primero conviértalo en el sprite en curso mediante TELL 1 y después digite EDSH. Entonces se visualizará la forma del sprite muy ampliada, y nos permitirá desplazar el cursor por la pantalla. Al pulsar la tecla del asterisco (*) se activará un pixel; si se pulsa la barra espaciadora se borrará.

Habiendo diseñado su sprite, pulse Control-C para definir la forma. Si el sprite no está visible, pruebe de ejecutar ST. Esta misma forma se le puede dar ahora también a otros sprites. SETSHAPE 1 (determinar forma) le proporcionará al sprite en curso la misma definición que al sprite 1. Después de haber definido un grupo de formas, puede guardar los sprites en un archivo con SAVESHAPES "NOMBREARCHIVO, y volver a leerlos con READSHAPES "NOMBREARCHIVO.

Hay un problema matemático muy conocido en el que se colocan cuatro insectos en las esquinas de un cuadrado. Todos parten a la misma velocidad y cada uno sigue al insecto que tiene a su derecha. El objetivo consiste en trazar sus recorridos. Aquí ofrecemos un programa en LOGO que implementa el problema utilizando sprites.

Los procedimientos que proporcionamos sitúan una copia del mismo sprite en cada esquina del cuadrado y luego los ponen en marcha siguiéndose los unos a los otros. La forma del insecto se define como el sprite 3 y a todos los otros se les da la misma forma utilizando SETSHAPE 3 en el procedimiento de posicionamiento.

El núcleo de nuestra solución está en el procedimiento SEGUIR. En éste, X e Y se establecen primero en las coordenadas *x* e *y* del sprite que se está siguiendo (:B) y luego el sprite que está haciendo el seguimiento (:A) tiene su encabezamiento orientado hacia ese punto. Para hacer esto, empleamos la primitiva TOWARDS (hacia). Ésta acepta dos parámetros de entrada, que representan las coordenadas del punto hacia el cual orientarse, y produce el encabezamiento del sprite en curso hacia ese punto.

Animación

Una interesante aplicación de los gráficos sprite es la creación de efectos de animación. Se define una serie de formas de sprite que representen el mismo objeto. Cada una de éstas es ligeramente diferente de la anterior, y cuando se ejecutan juntas crean un efecto de movimiento. El LOGO Commodore ofrece tres formas que crean la primaria figura de un hombre que corre. Los siguientes procedimientos preparan la pantalla y después ponen las tres formas en movimiento.

```
TO CORRER
   TELL 0
   DRAW
   PU
   BIGX BIGY
   SETH 90
   CORRIENDO 2
END
TO CORRIENDO :FORMA
   FD 5
   SETSHAPE :FORMA
   IF :FORMA=4 THEN MAKE "FORMA 1
   CORRIENDO :FORMA+1
END
```

Antes de ejecutar estos procedimientos, cargue el archivo SPRITES del disco de utilidades. Éste contiene unos cuantos procedimientos muy útiles, incluyendo BIGX y BIGY (grande), los cuales duplican el tamaño de un sprite. SMALLX y SMALLY (pequeño) son los procedimientos contrarios: se emplean para devolver un sprite a su tamaño original. Cargue los tres sprites digitando READSHAPES"RUNNER, y después ejecute los procedimientos.

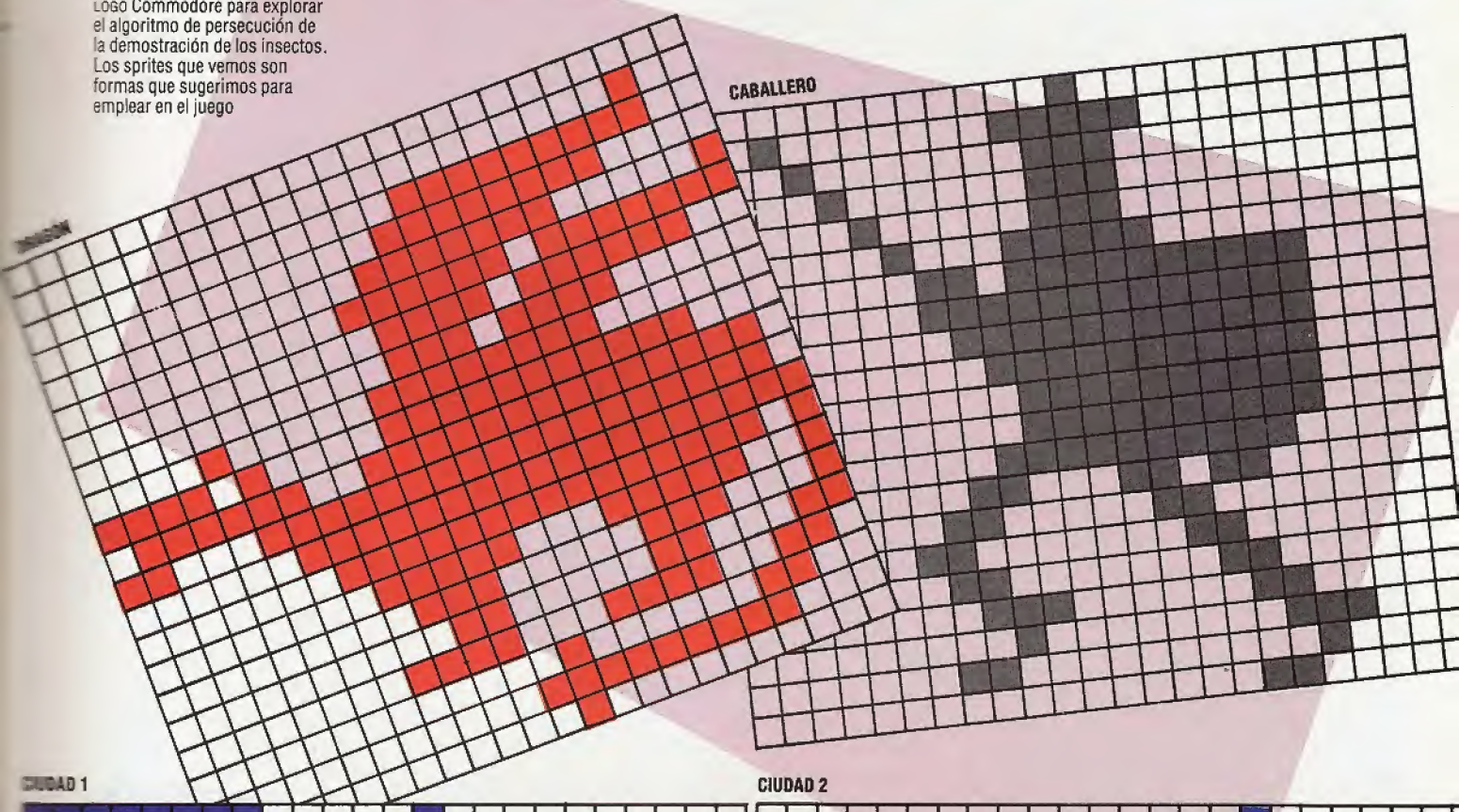
En la página siguiente también definimos cuatro formas de sprite, que usaremos luego en un juego.

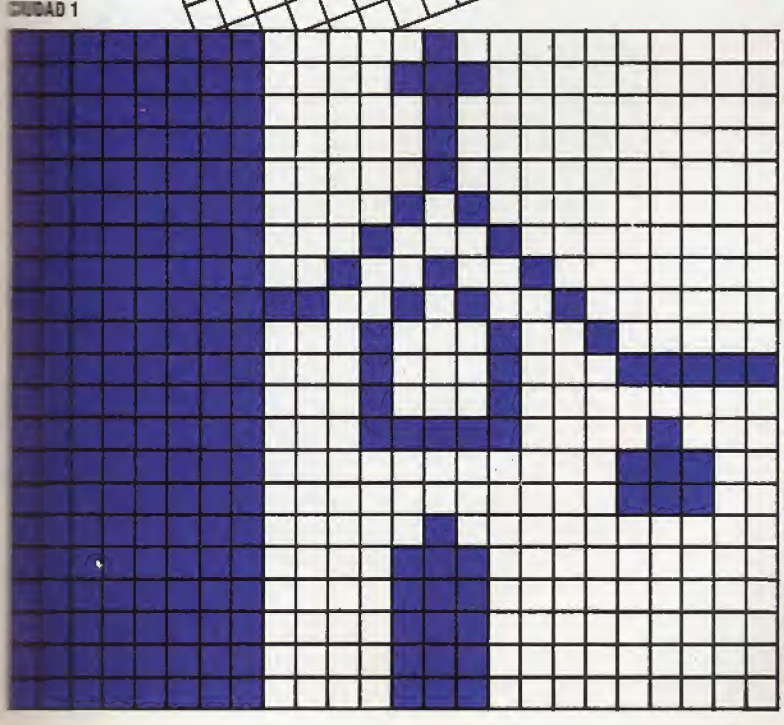
Complementos al LOGO

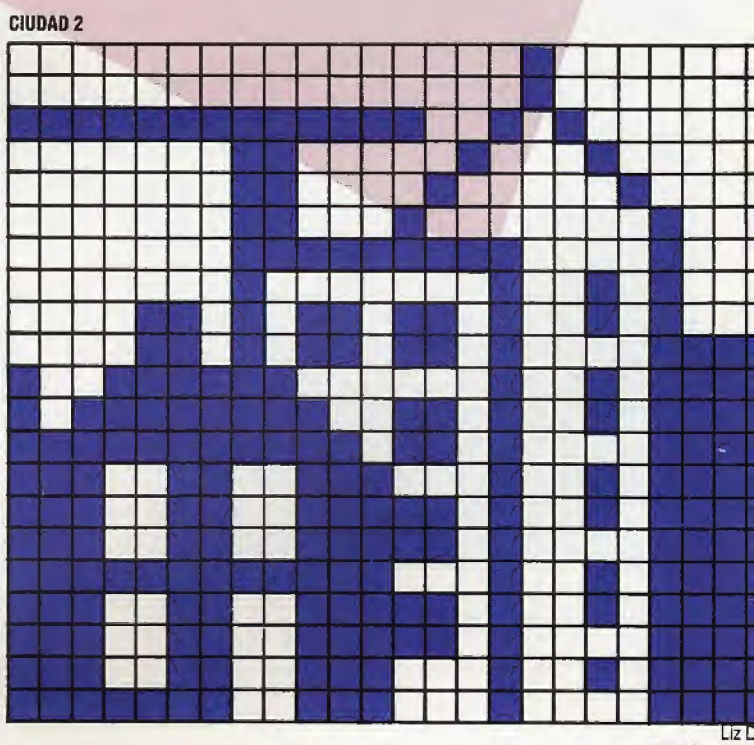
Ni el LOGO Spectrum ni el LOGO Apple poseen la facilidad de gráficos sprite. Los usuarios de Atari deben considerar estas diferencias:
1) Sólo hay cuatro sprites disponibles.
2) Para SETSHAPE utilizar SETSH.
3) El editor de sprites está incluido entre las primitivas. Pulsando la barra espaciadora se rellena un pixel vacío o se vacía uno lleno.

El dragón feroz

En el próximo capítulo publicaremos el juego del caballero y el dragón, que utiliza las facilidades de sprites del LOGO Commodore para explorar el algoritmo de persecución de la demostración de los insectos. Los sprites que vemos son formas que sugerimos para emplear en el juego



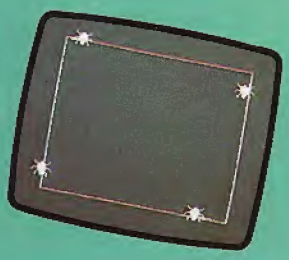


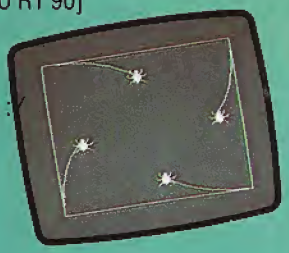


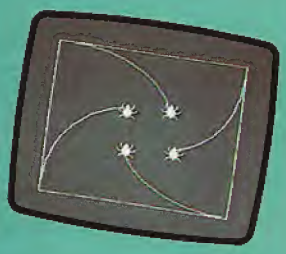
Cuatro insectos

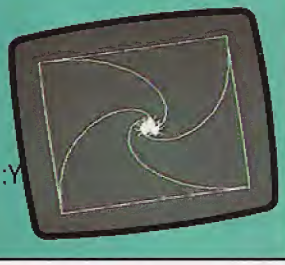
En esta demostración de geometría, cada insecto avanza directamente hacia la posición del insecto que tiene a su derecha. Este algoritmo produce una espiral hacia dentro; sus brazos describen la "curva de persecución" que tan familiar les resulta a los pilotos de combate y a los aficionados a los juegos recreativos

```
TO INSECTOS
   PREPARACION
   MOVER.INSECTOS
END

TO PREPARACION
   DRAW
   FULLSCREEN
   TELL 0
   HT
   PU
   SETXY(-100)(-100)
   CUADRADO 200
   POSICION 1(-100)(-100)
   POSICION 2(-100) 100
   POSICION 3 100 100
   POSICION 4 100(-100)
END

TO CUADRADO :LADO
   PD
   REPEAT 4[FD :LADO RT 90]
   PU
END

TO POSICION :NUM :X :Y
   TELL :NUM
   SETSHAPE 3
   PU
   SETXY :X :Y
   PD
   ST
END

TO MOVER.INSECTOS
   SEGUIR 1 2
   SEGUIR 2 3
   SEGUIR 3 4
   SEGUIR 4 1
   MOVER.INSECTOS
END

TO SEGUIR :A :B
   TELL :B
   MAKE"X XCOR
   MAKE"Y YCOR
   TELL :A
   SETH TOWARDS :X :Y
   FD 10
END
```









Proyecto Lunar Lander

Hemos desarrollado este programa como solución al propuesto en el capítulo anterior (p. 1135). Digite ALUNIZAR para participar en el juego:

```
TO ALUNIZAR
   PREPARACION JUEGO
END

TO PREPARACION
   DRAW DIBUJAR.PLATAFORMA
   PREPARAR.COHETE
END

TO DIBUJAR.PLATAFORMA
   PU SETXY(-20)(-60)PD SETXY 20(-60)PU
END

TO PREPARAR.COHETE
   SETXY 0 120 MAKE "VEL 0
   MAKE "COMBUSTIBLE 50
END

TO JUEGO
   ORDEN MOVER
   IF YCOR<-53 THEN BOOM STOP
   GRAVEDAD INFORME.COMBUSTIBLE JUEGO
END

TO ORDEN
   IF TECLALEIDA="F THEN QUEMAR
END

TO TECLALEIDA
   IF RC? THEN OUTPUT RC
   OUTPUT "
END

TO QUEMAR
   IF :COMBUSTIBLE>0 THEN MAKE "VEL
   :VEL+0.5 MAKE
   "COMBUSTIBLE :COMBUSTIBLE-1
END

TO MOVER
   SETY YCOR+:VEL
END

TO BOOM
   IF :VEL>(-1) THEN PRINT[HA HECHO UN
   BUEN ALUNIZAJE,FELICITACIONES] STOP
   PRINT [IMPACTO MATO A LA TRIPULACION!]
END

TO GRAVEDAD
   MAKE "VEL :VEL-0.2
END

TO INFORME.COMBUSTIBLE
   (PRINT"COMBUSTIBLE :COMBUSTIBLE)
END
```

Con tres insectos

Escriba un programa para otro problema, con tres insectos situados en las esquinas de un triángulo

Ian McKinnell

Promesa oriental

Varias empresas japonesas se han puesto de acuerdo en el proyecto MSX para producir un ordenador estándar

En esta ocasión analizaremos dos de los primeros microordenadores MSX fruto del proyecto MSX: el Sony Hit-Bit y el Toshiba HX-10.

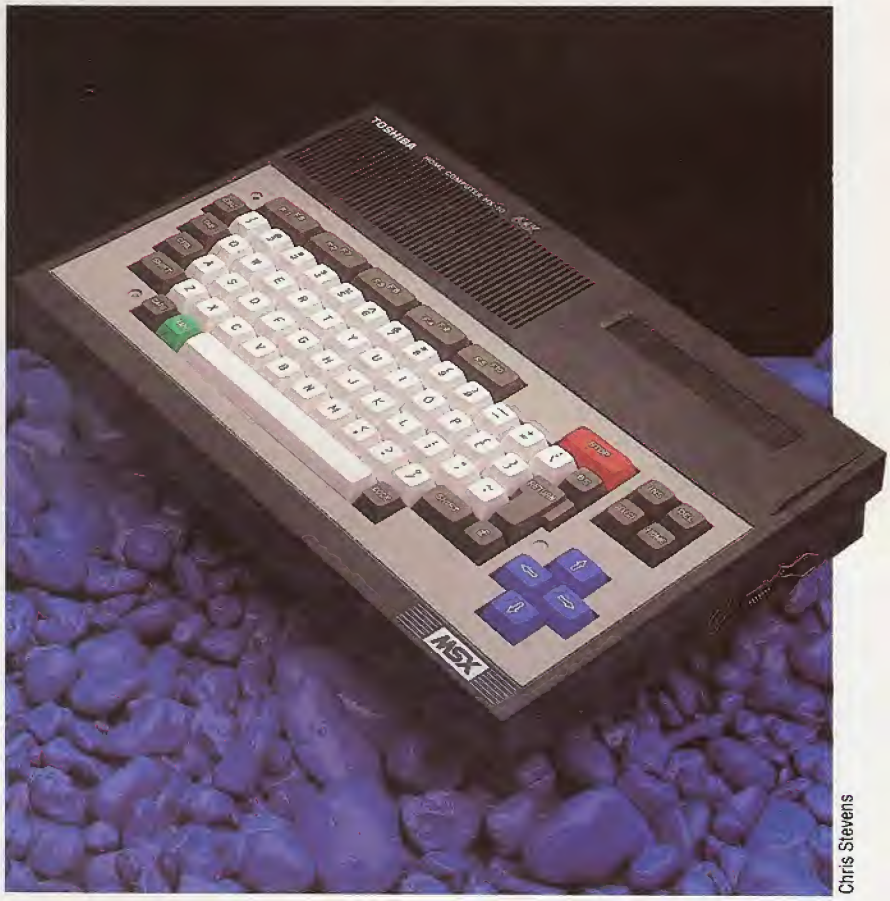
El estándar MSX (véase p. 621) determina la CPU que se utiliza (Z80), la cantidad mínima de ROM (32 Kbytes) y de RAM (8 Kbytes), el tipo de chips para gráficos y sonido, el contenido del teclado (si bien el trazado puede variar), la cantidad mínima de interfaces y su diseño; las pantallas para gráficos y para texto y, por supuesto, el lenguaje BASIC que contiene la ROM. Dado que el MSX es un diseño estandarizado, cabe esperar que todas las máquinas MSX serán similares. Los fabricantes tienen flexibilidad en cuanto a la cantidad de memoria por encima del mínimo, el tipo de teclado utilizado y la cantidad de interfaces extras. En la práctica, Sony y Toshiba, al igual que la mayoría de los fabricantes MSX, han optado por una especificación más amplia que la que dictan las exigencias mínimas.

Tanto el Sony Hit-Bit como el Toshiba HX-10 poseen teclados de buena calidad, si bien a algunos usuarios las teclas les resultarán demasiado sensibles. Los dos micros vienen con 64 Kbytes de memoria principal, y 16 Kbytes de RAM adicionales dedicados a la visualización en video. Esto da un total de 80 Kbytes, más de lo que ofrecen la mayoría de los ordenadores personales. Los modelos de Sony y Toshiba poseen cada uno una interface para impresora Centronics estándar y un par de conectores para palanca de mando, que en los ordenadores personales suelen ser opciones extras.

Originalmente se pensaba que los ordenadores MSX serían máquinas de precio reducido, pero las fluctuaciones monetarias y el aumento de los costos de producción han hecho subir los precios. Otra causa del aumento de precios ha sido la prisa por poner a la venta los ordenadores en Europa. Toshiba está vendiendo todas sus máquinas por envío aéreo, más costoso que el marítimo. La empresa ha tenido que modificar todas sus cadenas de producción, pasando de fabricar versiones japonesas de la máquina a construir el modelo europeo, con el deseo de convertirse en el primer fabricante MSX que tenga a la venta un producto en Europa.

Una de las primeras cosas que se observan al conectar un micro MSX es una fila de términos en la parte inferior de la pantalla. Éstas son palabras clave del lenguaje BASIC, como RUN, CLOAD", LIST, etc. Los micros tienen cinco teclas de función que generan estos términos tan comúnmente utilizados. Las palabras de la pantalla sirven como etiquetas para las teclas de función de modo que el usuario no tenga que recordar la función de cada tecla.

Estas teclas se definen automáticamente cuando se conecta la máquina, pero es fácil cambiar sus definiciones mediante el empleo de la instrucción KEY. A pesar de que sólo hay cinco teclas de función, se puede acceder a hasta 10 funciones pulsando la tecla Shift y la tecla de función deseada al mismo tiempo. Al pulsar Shift, las etiquetas de la pantalla cambian, pasando a reflejar las nuevas funciones asignadas a las teclas. Cada etiqueta de función puede contener hasta 15 caracteres, si bien sólo aparecerán en la pantalla los siete primeros.



Chris Stevens

Sistema estándar
El Toshiba HX-10 posee dos puertas para palanca de mando, una interface para impresora Centronics en paralelo, puerta para cartucho de ROM y "racimo" de teclas de manejo de cursor, como vemos en la fotografía. El BASIC MSX trata al control de la palanca de mando de la misma forma que el control mediante cursor, de modo que se pueden escribir juegos para un tipo de control y también utilizar automáticamente el otro tipo

El teclado y el editor de pantalla trabajan juntos para simplificar la edición. Cuatro teclas desplazan el cursor a través de la pantalla, y se pueden introducir cambios en cualquier lugar de ésta simplemente escribiendo sobre los caracteres existentes. Para insertar y eliminar caracteres se requiere la pulsación de una sola tecla. Las teclas para el cursor del Toshiba HX-10 son del mismo tamaño que las del resto del teclado, pero el Sony Hit-Bit utiliza teclas diferenciadas y grandes para su racimo de cursor. Éstas se emplean con mucha frecuencia, de modo que un diseño de esta clase puede resultar muy cómodo.

Al igual que el hardware, el software MSX está lleno de características extras. El BASIC MSX incluye instrucciones tales como AUTO y RENUMeración, y contiene varias instrucciones para generación de sonido, gráficos y tratamiento de interrupciones. Hay otras tres instrucciones fundamentales para crear gráficos. LINE dibuja una línea entre dos puntos, si bien se puede utilizar también para dibujar un cuadrado agregando la letra B (por *box*: caja) después de las coordenadas. Añadiendo las letras BF (por *box fill*: rellenar caja) se dibuja un cuadrado de color sólido. La instrucción CIRCLE se puede emplear para dibujar elipses y arcos además de círculos básicos. Y la instrucción PAINT rellenará con color sólido cualquier forma esbozada y funciona incluso con las formas más extrañas.

El BASIC MSX incluye muchas otras características útiles, aunque tal vez el juego de instrucciones más impresionante (para tratamiento de interrupciones) no se aprecie al principio. El tratamiento de interrupciones es muy útil en la programación de gráficos de alta velocidad. Existe una gran cantidad de situaciones en las que un programa debe realizar una tarea, verificando constantemente al mismo tiempo si sucede alguna otra cosa. Un ejemplo típico de esto se puede encontrar en juegos del tipo "marcianitos". El programa debe mantener a los

Enarbolando la bandera

El estándar MSX

CPU	Z80A, 3,58 MHz
RAM	Mínimo: 8 K
ROM	32 K incluyendo BASIC
PANTALLA	16 colores, gráficos de 256×192 pixels, 32 sprites, visualización de texto de 40×24 (o 32×24) (chip de video TI 9918 o equivalente)
SONIDO	3 canales, accesibles desde BASIC (chip controlador de sonido AY38910)
INTERFACES	Puerta para cartucho MSX, salida modulada para TV, impresora en paralelo Centronics, interface para cassette
TECLADO	Teclado QWERTY más teclas de función especial, 4 teclas de cursor, 10 teclas de función programables

Complementos al MSX

SONY HIT-BIT	Software de base de datos incorporado, salida RGB, paquetes opcionales RAM 4 K
TOSHIBA HX-10	Bus de ampliación, 2 puertas para palanca de mando
YAMAHA CX-5	Teclado minimúsica y software con MIDI
PIONEER	Interface controladora de videodisco
SANYO MPC100	Lápiz óptico y software opcional
JVC HC7GB	Salida RGB
SPECTRAVIDEO SVI 728	Teclado numérico completo

Si bien el estándar MSX determina un mínimo de 8 K de memoria, todos los fabricantes mencionados proporcionan 64 K de RAM para el usuario, más 16 K de RAM de pantalla

extraterrestres moviéndose por la pantalla, verificando en todo momento si se ha pulsado o no el botón de "disparo". El programa debe hacer dos cosas al mismo tiempo, pasando rápidamente de una tarea a otra.

La solución MSX consiste en designar ciertas cosas como *eventos*. Se dispone de instrucciones para decirle al ordenador que vea si ha ocurrido un evento. Cuando se produce uno, el ordenador pasa automáticamente a una subrutina para tratar el evento.

La pantalla de gráficos MSX puede visualizar 16 colores con una resolución de 256×192 pixels. Se pueden definir hasta 32 sprites de 8×8 pixels (o 16 sprites de 16×16, u 8 sprites de 32×32). Para aprovechar los sprites al máximo, el BASIC MSX incluye un juego completo de instrucciones específicas para ello, como SPRITE, para definir un sprite, y PUT SPRITE, para situar un sprite en cualquier lugar de la pantalla.

Tal como han afirmado los fabricantes MSX, ya hay muchísimo software en cartucho a la venta para las máquinas. Y la promesa de la compatibilidad parece ser verdadera: el software para el Toshiba HX-10 funciona perfectamente con el Sony Hit-Bit, y viceversa. Esto se aplica tanto al software en cartucho como a los programas en cassette. Después de años y años de sistema incompatibles, parece casi mágico sacar un cartucho de una máquina y utilizarlo en otra. Las empresas MSX se están apoyando en esta característica para sacar muy rápidamente a la venta una amplia gama de software para todas las máquinas.

Queda por ver si las máquinas MSX tendrán en el mercado el impacto que esperan los japoneses. Con la fuerte competencia encabezada por Sinclair, Commodore y Amstrad, entre otras, se avecina una enconada contienda por las ventas. Sea como fuere, las máquinas MSX responden cabalmente a las afirmaciones de sus fabricantes. Son ordenadores agradables de utilizar, están bien equipados y tienen un precio razonable.

Standard Interface

Chris Stevens

TOSHIBA HX-10 MSX

DIMENSIONES
365×245×60 mm

CPU
Z80A, 3,58 MHz

MEMORIA
64 K de RAM (28 K disponibles para BASIC), 16 K de RAM de pantalla, 32 K de ROM incluyendo BASIC

PANTALLA
40 columnas por 24 filas para texto, 256×192 para gráficos, con 16 colores y hasta 32 sprites

INTERFACES
Impresora Centronics, TV, pantalla, salida de audio, 2 puertas para palanca de mando, puerta para cassette, ranura para cartucho de ROM, bus de ampliación

LENGUAJES DISPONIBLES
Microsoft Extended BASIC

TECLADO
68 teclas tipo máquina de escribir, con racimo de cursor más 5 teclas de función programables

DOCUMENTACION
Guía de instalación y guía de referencia para programación en BASIC. Ambas están bien hechas, pero no son lo suficientemente descriptivas

VENTAJAS
El BASIC MSX posee muchas características útiles, incluyendo buenas instrucciones para gráficos y sonido. La estandarización del MSX es un dato valioso porque supondrá más software y periféricos

DESVENTAJAS
El BASIC MSX carece de la capacidad de programación estructurada; la disponibilidad de MSX y los periféricos está tardando mucho en aparecer

DIFERENCIAS
El Sony Hit-Bit tiene 3 programas incorporados en ROM, salida para pantalla RGB, y la disposición del teclado es ligeramente distinta

Control total

En este último capítulo sobre software integrado analizaremos algunas alternativas al programa único multifunción y de gran consumo de memoria

El enfoque alternativo al software integrado responde a un principio completamente diferente. Se basa en que el sistema operativo del ordenador proporcione las facilidades básicas de integración y que los programas individuales escritos para trabajar con ese sistema automáticamente se adapten y trabajen juntos.

Cumpliendo órdenes
Con el sistema operativo tradicional, el programa que está en ejecución tiene el mando absoluto. Su lógica determina lo que aparece en la pantalla, cuándo se ha de acceder a la unidad de disco y cuándo leer del teclado. Sus instrucciones generales pasan al sistema operativo, que administra en detalle el empleo del hardware que se está usando. El programa en ejecución es el soberano, y se da por sentada la subordinación del sistema operativo

La creación de un sistema operativo de estas características no ha sido tarea fácil, puesto que exige que el hardware y el software del ordenador sean más sofisticados que el de los diseños tradicionales. Apple ha abierto el camino con sus ordenadores Lisa y Macintosh, diseñados para el cliente, si bien hay otras empresas, en particular Microsoft, que están preparando sistemas para otros ordenadores populares, como el IBM PC.

Los programas para estos nuevos sistemas operativos son muy distintos de los programas para sistemas tradicionales. Gran número de los programas se dedican a la interface para el usuario: las rutinas que reciben órdenes e información del usuario y presentan los resultados. Las opiniones difieren en cuanto a la operatoria de los programas, de modo que casi todos los paquetes poseen sus propios procedimientos y han de ser aprendidos desde cero.

Un sistema operativo integrado proporciona un juego incorporado de rutinas de interface para el usuario para que lo utilicen todos los programas de aplicaciones. Cuando un programa desea visualizar una lista de opciones en la pantalla para que el usuario realice su elección entre ellas, debe usar para ello una rutina ya incorporada en el sistema operativo. La ventaja que esto ofrece reside en que todos los programas escritos para trabajar con el mismo sistema operativo tendrán prácticamente los mismos procedimientos de operatoria. Una vez que el usuario ha aprendido a utilizar un programa en el sistema, ¡está listo para emplear el resto de programas disponibles!

Una interface para el usuario proporcionada en particular para estos programas es el ratón. Éste es un dispositivo señalador que se emplea para seleccionar opciones de la pantalla a través de un cursor no tradicional. Una alternativa es la *pantalla sensible al tacto*, en la que una matriz de haces luminosos responde ante el toque de un dedo. La visualización se divide en "ventanas" separadas, cada una de las cuales contiene una opción o tarea diferente. Técnicamente, una interface para el usuario de este tipo requiere un procesador rápido, muchísima memoria y gráficos de gran resolución. Pero estos costos adicionales bien valen la pena, porque el sistema, por lo general, es aplicable a casi cualquiera de los programas disponibles, es muy fácil de aprender y proporciona la forma más sencilla posible de que el usuario sea capaz de ver varias aplicaciones a la vez y pasar de una a otra.

Control de operaciones

Es importante apreciar la forma en que este sistema integra los programas. El programa y el usuario no están nunca en contacto directo: todo se ha de hacer a través del sistema operativo, y es éste el que tiene el control en todo momento. De hecho, cada programa de aplicaciones se convierte en una extensión del sistema operativo y el ordenador es un único "entorno" integrado.

Esto nos lleva a la segunda diferencia fundamental en la forma en que funcionan esta clase de sistemas. En un sistema tradicional, la comunicación entre programa y sistema operativo es en gran parte unidireccional. El programa solicita que se lleve a cabo una tarea específica y el sistema operativo la realiza en consecuencia.

En un sistema integrado, el sistema operativo está al mando y a él le corresponde solicitar cosas al programa. Por ejemplo, el sistema operativo puede enviarle al programa un mensaje que diga "¿Podrías volver a dibujar tu visualización, ya que el usuario la ha desplazado a otra zona de la pantalla?"; o "Manténlo todo; el usuario ha llevado el ratón a una aplicación distinta"; o "Aquí hay algunos datos para ti tomados de una hoja electrónica". O sea, el programa ha de ser capaz de responder a las exigencias y las demandas del sistema operativo, contrariamente a los sistemas tradicionales.

Una vez que se tiene este grado de cooperación entre todo el software de una máquina, es fácil construir un entorno integrado. Cada programa posee su propia ventana en la pantalla. Cuando el usuario coloca el ratón dentro de la ventana y selecciona una opción, el sistema operativo lo notifica a este programa determinado y se lleva a cabo la operación correspondiente.

Por ejemplo, si el usuario se mueve hasta la esquina de la ventana y selecciona la opción para seleccionar esa ventana y desplazarla a una nueva posición, las rutinas del sistema operativo llevan a cabo la tarea y luego, de ser necesario, informan al programa sobre los cambios, de modo que pueda modificar su visualización adecuadamente. Si el usuario lleva al ratón hasta una ventana diferente, el programa original queda temporalmente suspendido y el sistema operativo comienza a trabajar con el nuevo programa: pasar de una aplicación a otra es tan sencillo como desplazar el ratón.

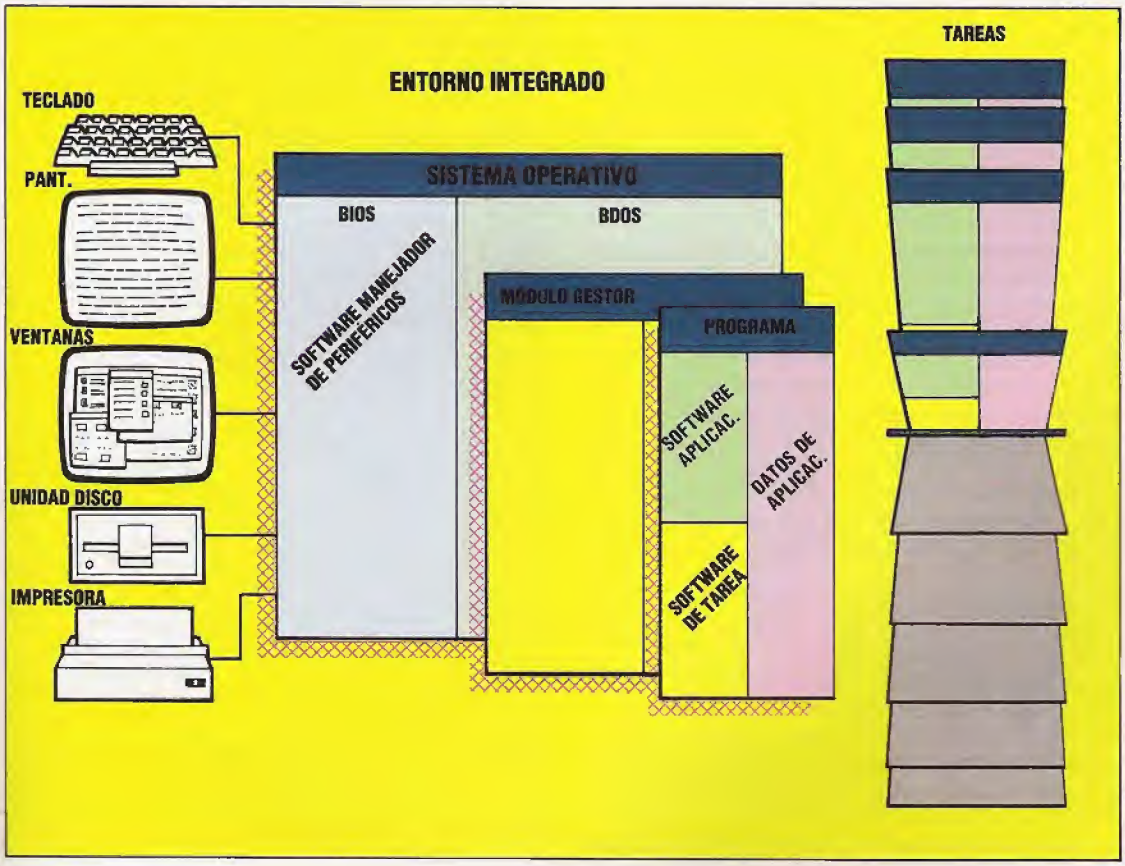
Al igual que los grandes programas "todo en uno", tales sistemas requieren que los programas y la información que sale en la pantalla en un momento dado estén en la memoria y listos para utilizar. Para facilitar esto, muchos sistemas poseen memorias masivas: un Megabyte en el Apple Lisa, por ejemplo, y 512 Kbytes en el Macintosh. Aun así, por lo general es necesario que el sistema operativo ocasionalmente deba intercambiar información y programas entre memoria y discos para acomodar todo. Para conseguir una rapidez aceptable, suele ser necesario operar en un disco rígido.

Con el objeto de que los datos se puedan intercambiar fácilmente entre los programas, el sistema operativo posee un conjunto de formatos y rutinas incorporado para transferir información. Cuando se "exportan" algunos datos de un programa y se requiere "importarlos" a otro, el sistema operativo suspende el primer programa e inicia el segundo; luego le solicita a la aplicación en curso que lea y procese la información proveniente de otro programa. Estos caminos se pueden preparar automáticamente, de modo que cuando se modifica la información de una hoja electrónica, por ejemplo, también se modificará de manera automática una gráfica de la misma hoja electrónica. Los dos programas no se ejecutarán al mismo tiempo: el sistema operativo simplemente hace malabarismos entre los dos en la medida de lo necesario.

El Apple, de Lisa, nos presenta una opción ligeramente más sofisticada, en que la información puede ser reducida a una ventana tipo "tablilla de anuncios" desde cualquier programa y luego ser "pegada" en otro. La información de formateo es transportada junto con los datos, de modo que un gráfico creado con software de gráficos de gestión es transferido como tal a otro programa.

Ésta es, por tanto, la forma más acertada de crear software integrado. Permite que el usuario mezcle y empareje cualesquiera de los programas del sistema, pase de uno a otro y transfiera información entre ellos con facilidad. El inconveniente es que requiere un hardware sofisticado que, por el momento, es bastante caro y que hay muy poco software disponible para integrar.

No obstante, toda innovación tecnológica de esta magnitud llevará tiempo antes de convertirse en algo cotidiano. El ratón y la interface para ventanas, por ejemplo, los desarrollaron equipos de investigación de Xerox hace ya más de diez años, ¡pero ha sido sólo ahora cuando tal sistema ha podido ponerse a la venta en las tiendas!



Kevin Jones

Operaciones combinadas
En un sistema integrado, el sistema operativo se ve mejorado por la adición de un módulo "gestor", que trata a todos los programas y datos en curso como "tareas" a planificar y procesar y manipula el detallado sistema operativo subyacente como un simple software de apoyo al sistema. Este módulo coloca y elimina las tareas en la memoria principal y en los discos de acuerdo a los requerimientos del usuario y las necesidades de las tareas en curso. Está equipado para pasar la información entre las aplicaciones en formatos estándares y, por tanto, posibilita la transferencia de datos entre las tareas. De hecho, el gestor es en sí mismo una tarea de alta prioridad, y su relación con las otras tareas es simbiótica en vez de servil

Dando la alarma

He aquí algunos programas para utilizar la caja de relés en sencillas aplicaciones domésticas

La caja de relés está diseñada para controlar el suministro de electricidad a cualquier dispositivo que esté conectado a ella. En respuesta a una señal de bajo voltaje, la caja abre o cierra la alimentación de corriente de la red al conector montado en la caja. El modo de operación es tal que la alimentación de la red al conector se mantiene mientras haya una corriente de poco voltaje suministrada al relé. Por consiguiente, podemos activar éste directamente desde la caja de salida de bajo voltaje que construimos previamente (véase p. 1054). El suministro de energía eléctrica de la red desde la caja de relés reflejará exactamente la corriente de bajo voltaje proporcionada al relé desde la caja de salida de bajo voltaje. Por lo tanto, se puede conseguir el control de la alimentación de red mediante las mismas técnicas de software utilizadas para controlar los dispositivos de bajo voltaje.

Si, por ejemplo, se conectan los cables de bajo voltaje del relé a las conexiones positiva y negativa de la línea 0 de la caja de salida, y se enchufa en un conector de corriente, se le suministrará corriente al conector de la caja de relés cuando el bit 0 del registro de datos de la puerta para el usuario se envíe alto. Siempre que el bit 0 se envíe bajo se interrumpirá el suministro de electricidad al conector de la caja de relés. Hasta cuatro cajas de relés o disyuntoras se pueden conectar a la caja de salida de bajo voltaje y conmutar de este modo.

Podemos valernos de esta sencilla disposición de conmutación para desarrollar unos cuantos sistemas de control que utilicen los aparatos domésticos de uso cotidiano. Primero probaremos un proyecto sencillo, en el cual utilizamos una grabadora de cinta para programar un micro de modo que responda "verbalmente" a la presión sobre un teclado.

En primer lugar, necesitamos grabar una serie de frases, tales como "Estás pisoteando mi teclado", seguido de "Lo has vuelto a hacer" y "Oye, ¡te lo he advertido!", y así sucesivamente. Después de grabados los mensajes, conectaremos el teclado y la grabadora al sistema de la puerta para el usuario y escribiremos un poco de software para activar las frases, de una en una, en respuesta a una presión repetida sobre el teclado.

En el sistema de la puerta para el usuario hemos de hacer las siguientes conexiones:

1) Enchufar los cables de voltaje del relé o disyuntor de red en los terminales positivo y negativo de la línea 0 de la caja de salida de bajo voltaje.
2) Enchufar el cable de alimentación a la caja de relés en un enchufe de pared.
3) Conectar los dos cables del teclado de presión a través de los terminales positivo y negativo de la línea 7 de la caja buffer.

El principal problema que supone el diseño de software para este sistema es asegurar que la grabadora se encienda y se apague con exactitud cuando se reproduce un mensaje. Por consiguiente, antes de que podamos escribir un programa debemos cronometrar con suma precisión cada mensaje e introducir estos datos en el programa controlador. El cronometraje se puede realizar utilizando el reloj interno del micro o un cronómetro. Si en la cinta hay tres frases que duran períodos de T(1), T(2) y T(3) segundos, entonces podemos escribir un programa que, al activarse desde el teclado de presión, encienda la grabadora de cinta durante el período de tiempo correcto para cada uno de los sucesivos mensajes. Si el cronometraje de las frases se efectúa con precisión, entonces cada frase estará justo en su inicio cuando se encienda la grabadora.

Los siguientes programas (para el Commodore 64 y el BBC Micro) activarán la grabadora para tres intervalos de tiempo sucesivos T(1), T(2) y T(3) en respuesta a impulsos provenientes del teclado de presión. Estas variables deben ser inicializadas con los valores correspondientes a las tres frases elegidas.

BBC Micro

```
 10 REM PROGRAMA PISOTEANDO BBC
 20 DIM T(3)
 30 RDD=&FE62:REGDAT=&FE60
 40 ?RDD=127:REM L7 ENTRADA
 50 ?REGDAT=0:REM TODAS APAGADAS
 60 CLS
 70 FORI=1TO3
 80 INPUT"INTERVALO DE TIEMPO (SEGS)";T(I)
 90 NEXT I
100 :
110 FORL=1TO3
120 CLS
130 REPEAT
140 UNTIL(?REGDAT AND 128)=0:REM L7 BAJA
150 ?REGDAT=1:REM CONECTAR CINTA
160 TIME=0:REM INICIAR RELOJ
170 REPEAT
180 UNTIL TIME>T(L)*100
190 ?REGDAT=0:REM APAGAR CINTA
200 NEXT L
210 END
```

Commodore 64

```
 10 REM PROGRAMA PISOTEANDO CBM 64
 20 DD=56579:REGDAT=56577
 30 POKERDD,127:REM L7 ENTRADA
 40 POKEREGDAT,0:REM TODAS APAGADAS
 50 PRINTCHR$(147): REM LIMPIAR PANTALLA
 60 FOR I=1 TO 3
 70 INPUT"INTERVALO DE TIEMPO (SEGS)";T(I)
 80 NEXT I
 90 :
100 FOR L=1TO3
110 IF(PEEK(REGDAT)AND128)<>0 THEN 110
115 POKE REGDAT,1:REM ENCENDER CINTA
120 T=TI:REM INICIALIZAR RELOJ
130 IF T(L)>(TI-T)/60 THEN130
140 POKEREGDAT,0:REM APAGAR CINTA
150 NEXT L
160 END
```

Reloj despertador

Habiendo desarrollado un sistema sensible a las pisadas sobre un teclado de intrusos, vamos ahora a considerar un proyecto para convertir un micro en un cómodo reloj despertador programable. Un sistema de este tipo se puede, por supuesto, confeccionar a la medida exacta de las necesidades de cada uno. El programa que ofrecemos (en versiones para el Commodore 64 y el BBC Micro) permite que el usuario entre:

1) la hora del día;
2) el número de intervalos "para cabecear" (períodos entre los estallidos del zumbador o música) requeridos;
3) si para cada intervalo para cabecear se requiere un período de música, de alarma o de silencio, y la longitud del intervalo;
4) si para cada intervalo se debe encender o no una luz;
5) la última hora posible para levantarse.

El programa se basa en la presunción de que se realizan las siguientes conexiones a la caja de salida de bajo voltaje:

1) Se conecta una grabadora de cinta a la línea 0 a través de un relé o disyuntor de red.
2) Se conecta una lamparilla de mesa a la línea 1 a través de un relé en la línea 1.
3) Se conecta directamente a la línea 3 un timbre eléctrico de 9 voltios.

El programa acepta la última hora para levantarse y cuenta hacia atrás el número de intervalos programados para calcular la hora de comienzo de cada intervalo. Se utilizan matrices para almacenar los datos que nos dicen qué aparatos han de estar encendidos durante cada período. Observe que a las variables de matriz se les asignan valores que corresponden al valor de bits requerido en el registro de datos para encender cada aparato en particular. Mediante el empleo de la instrucción lógica OR podemos simplemente hallar el total compuesto que se debe colocar en el registro de datos para activar cualquier combinación de dispositivos.

La mayor parte de nuestro esfuerzo de programación se ha dirigido hacia la manipulación de variables en serie (*strings*) para permitir realizar cálculos numéricos. Esto es especialmente cierto para el programa del Commodore 64, porque la versión de BASIC que utiliza esta máquina carece de las útiles instrucciones MOD y DIV de que disponen los programadores del BBC Micro.

Ahora hemos desarrollado un sistema de entrada y salida verdaderamente flexible para control por microordenador, que nos permite controlar LED, dispositivos de bajo voltaje y aparatos que funcionan con la red eléctrica, además de permitir que el micro acepte e interprete datos entrados desde una gama de sensores. Ahora se nos abren muchas posibilidades en el diseño de sistemas de control para que los empleemos nosotros mismos. En los ejemplos que ofrecemos el micro se utiliza como un sofisticado reloj programable. Otras aplicaciones implicarían encender y apagar estufas eléctricas en respuesta a un par de sensores de calor, o encender una bombilla eléctrica por la noche. Las posibilidades para la experimentación son infinitas.

Ian McKinnell

Commodore 64

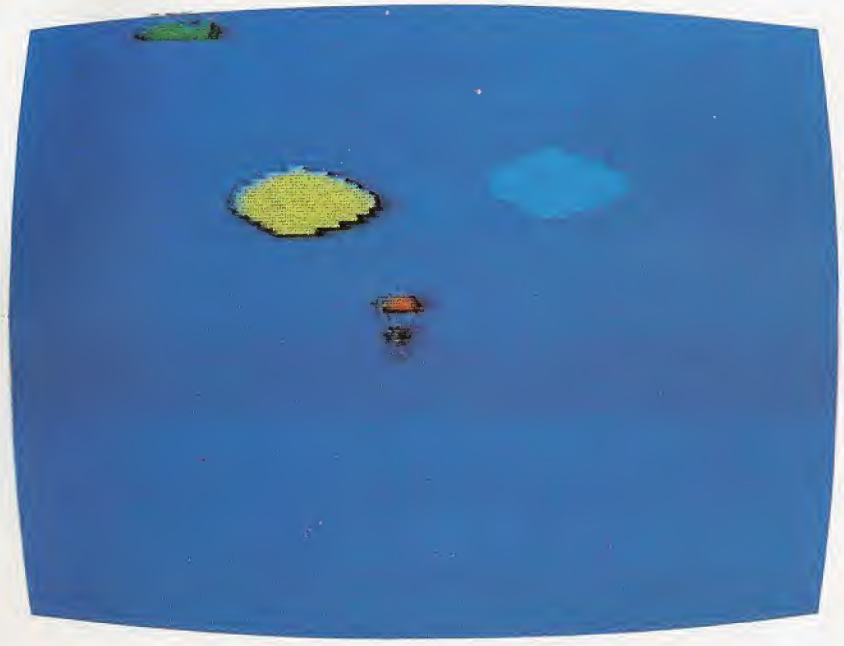
```
100 REM**** RELOJ DESPERTADOR CBM 64 ****
110 RDD=56579:REGDAT=56577
120 POKE RDD,255:POKEREGDAT,0
130 PRINTCHR$(147):REM LIMPIAR PANTALLA
140 INPUT"NUMERO DE INTERVALOS PARA CABECEAR";N
150 M=N+1
160 DIM A(M),M(M),L(M),TS$(M),T(M)
170 :
180 REM**** ENTRAR DATOS INTERVALOS ****
190 FOR C=1 TO N
200 PRINT:PRINT"NUMERO DE INTERVALO";C
210 INPUT"MUSICA,ALARMA O SILENCIO (M/A/S)";AN$
215 AN$=LEFT$(AN$,1)
220 IF AN$<>"M"ANDAN$<>"A"ANDAN$<>"S" THEN 210
230 IF AN$="M" THEN M(C)=1:A(C)=0
240 IF AN$="A" THEN A(C)=8:M(C)0
250 IF AN$="S" THEN A(C)=0:M(C)=0
260 INPUT"LUZ ENCENDIDA (S/N)";AL$
270 L$=LEFT$(AL$,1)
280 IF L$<>"S" AND L$<>"N" THEN 260
290 IF L$="S" THEN L(C)=2:GOTO310
300 L(C)=0
310 INPUT"INTERVALO DE TIEMPO (MIN)";T(C)
320 NEXT C
330 :
340 INPUT"HORA MAXIMA PARA LEVANTARSE (HHMM)";LT$
350 LT$=LT$+"00":REM AGREGAR SEGUNDOS
360 TS$(N+1)=LT$:REM HORA MAXIMA
370 REM CONVERTIR HORA MAXIMA A MINUTOS
380 LM=60*VAL(LEFT$(LT$,2))+VAL(MID$(LT$,3,2))
390 :
400 INPUT"AHORA SON LAS (HHMM) ";TN$
410 TI$=TN$+"00": REM INICIAR RELOJ
420 :
430 REM**** ANALIZAR Y CALCULAR ****
440 REM ****** CALCULAR HORAS COMIENZO INTERVALOS ******
450 FOR C=N TO 1 STEP -1
460 LM=LM-T(C):REM HORA COMIENZO EN MIN
470 HR=INT(LM/60)
480 MN=INT(60*(LM/60-HR+.000001))
490 HR$=STR$(HR):REM HORAS
500 MN$=STR$(MN):REM MINUTOS
510 MN$=MID$(MN$,2,LEN(MN$))
520 HR$=MID$(HR$,2,LEN(HR$))
530 REM** AGREGAR CEROS POR DELANTE **
540 SP$="00"
550 HR$=LEFT$(SP$,2-LEN(HR$))+HR$
560 MN$=LEFT$(SP$,2-LEN(MN$))+MN$
570 TS$(C)=HR$+MN$+"00"
580 NEXT C
590 :
600 REM** LISTO **
610 PRINTCHR$(147)
620 FOR C=1 TO N+1
630 IF TI$<TS$(C)THENGOSUB710:GOTO630
640 DN=M(C) OR A(C) OR L(C):REM DATOS REGDAT
650 POKE REGDAT,DN
670 NEXT C
680 POKE REGDAT,0
690 END
700 :
710 REM**** S/R VISUALIZAR RELOJ ****
720 PRINTCHR$"145);:REM CURSOR ARRIBA
730 PRINTLEFT$(TI$,2);":";MID$(TI$,3,2);
740 PRINT"  ";RIGHT$(TI$,2)
750 RETURN
```

BBC Micro

```
10 REM RELOJ DESPERTADOR BBC
15 MODE7
20 RDD=&FE62:REGDAT=&FE60
30 CLS
40 INPUT"CANTIDAD DE INTERVALOS PARA CABECEAR";N
45 M=N+1
50 DIM A(M),M(M),L(M),T(M),TS(M)
70 REM**** INPUT DATOS INTERVALOS ****
80 FORC=1 TO N
90   PRINT"NUMERO DE INTERVALO";C
95   REPEAT
100    PRINT "MUSICA,ALARMA O SILENCIO";
103    INPUT" (M/A/S)";AN$
105    AN$=LEFT$(AN$,1)
110    UNTIL AN$="M"ORAN$="A"ORAN$="S"
120  IF AN$="M"THEN M(C)=1:A(C)=0
130  IF AN$="A" THEN M(C)=0:A(C)=8
140  IF AN$="S" THEN M(C)=0:A(C)=0
150  REPEAT
160    INPUT"LUZ ENCENDIDA (S/N) ";AL$
170    AL$=LEFT$(AL$,1)
180    UNTIL AL$="S" OR AL$="N"
190  IF AL$="S" THEN L(C)=2 ELSE L(C)=0
200  INPUT"INTERVALO TIEMPO (MIN)";T(C)
210  NEXT C
220 :
230 INPUT"HORA MAXIMA PARA LEVANTARSE (HHMM) ";LT$
232 TS(N+1)=6000*(60*VAL(LEFT$(LT$,2)))
234 TS(N+1)=TS(N+1)+VAL(RIGHT$(LT$,2))
236 REM CONVERTIR HORA MAXIMA A MINUTOS
237 LM=60*VAL(LEFT$(LT$,2))
239 LM=LM+VAL(RIGHT$(LT$,2))
240 INPUT"AHORA SON LAS (HHMM)";TN$
250 TIME=6000*(60*VAL(LEFT$(TN$,2)))
255 TIME=TIME+VAL(RIGHT$(TN$,2))
260 :
270 REM ANALIZAR Y CALCULAR
280 FORC=N TO 1 STEP -1
290   LM=LM-T(C):REM COMIENZO INTERVALOS
300   TS(C)=6000*LM
310   NEXT C
320 :
330   REM**** LISTO ****
340 CLS
350 FOR C=1 TO N+1
360   REPEAT
370     PROCreloj
380     UNTIL TIME>=TS(C)
390   DATOSREG=M(C)OR A(C) OR L(C)
400   ?REGDAT=DATOSREG
420   NEXT C
430 ?REGDAT=0
440 END
999 :
1000 DEF PROCreloj
1020 MIN=(TIME DIV 6000) MOD 60
1030 HR=(TIME DIV 6000) MOD 60
1040 MIN$=STR$(MIN):HR$=STR$(HR)
1042 REM AGREGAR CEROS POR DELANTE
1043 SP$="00"
1044 HR$=LEFT$(SP$,2-LEN(HR$))+HR$
1045 MIN$=LEFT$(SP$,2-LEN(MIN$))+MIN$
1050 PRINTTAB(18,12)HR$;":";MIN$
1060 ENDPROC
```

Paracaídas

Iniciamos un nuevo apartado, en el que proporcionaremos listados de juegos. He aquí "Paracaídas", para el Commodore 64



Saltando de un helicóptero en vuelo, intente alcanzar el blanco situado en el suelo. Una primera presión sobre una tecla le permitirá bajar verticalmente en caída libre. Una segunda presión abrirá el paracaídas. El descenso continuará más lentamente, en un ángulo de 45°, por el empuje del viento. Cuanto más espere a abrir el paracaídas, menor será la desviación. Pero no aguarde demasiado, ya que por debajo de los 100 m éste no se abrirá.

```
  5 REM**********************************************
 10 REM*                  PARACAIDAS                  *
 15 REM**********************************************
 20 PRINT CHR$(147)
 25 GOSUB 10000
 30 POKE V+5,50
 40 POKE V+1,100
 50 POKE V+3,95
 70 AV=320
 80 N2=320
 85 N1=0
 90 MS=6
100 POKE 2040,14
110 POKE 2041,14
115 POKE 2042,13
120 POKE 2043,15
125 POKE 2044,16
130 POKE 2045,17
135 POKE V+23,3
140 POKE V+29,3
150 POKE V+39,1
160 POKE V+40,14
170 POKE V+41,5
200 POKE V+42,10
210 POKE V+43,10
215 POKE V+44,5
220 VV=4
230 NN=0
240 SA=0
450 POKE V,N1
460 POKE V+2,63
470 POKE V+4,63
480 POKE V+16,MS
490 POKE V+21,39
500 AV=AV-2
510 IF AV<1 THEN AV=320:MS=MS+4
520 IF AV=254 THEN MS=MS-4
540 N1=N1+1
550 IF N1>320 THEN N1=0:MS=MS-1
560 IF N1=256 THEN MS=MS+1
580 N2=N2-1
590 IF N2<1 THEN N2=320:MS=MS+2
600 IF N2=255 THEN MS=MS-2
610 H=H+VV
620 IF H>255 THEN H=255
630 IF H=230 AND SA>0 THEN 5000
1000 POKE V+16,MS
1010 IF N1<256 THEN POKE V,N1:GOTO 1030
1020 POKE 2043,15
1025 POKE 2044,16
1030 IF N2<256 THEN POKE V+2,N2:GOTO 1050
1040 POKE V+2,N2-255
1050 IF AV<256 THEN POKE V+4,AV:GOTO 1070
1060 POKE V+4,AV-255
1070 GET X$
1080 IF X$="" OR AV>255 THEN 1500
1090 ON SA GOTO 1300,1500
1100 SA=1
1110 P=AV
1120 POKE V+6,P
1130 POKE V+21,47
1140 H=58
1150 GOTO 1500
1300 IF H>150 THEN 1500
1305 POKE V+21,55
1310 VV=1
1320 NN=1
1340 SA=2
1500 P=P-NN
1510 IF P<1 AND SA>0 THEN 6000
2000 POKE V+7,H
2010 POKE V+9,H
2020 POKE V+6,P
2030 POKE V+8,P
2040 GOTO 500
5000 IF ABS(L-P)>4 OR SA=1 THEN 6000
5010 SC=SC+10
5020 FOR I=1 TO 1000
5030 NEXT I
5040 RESTORE
5050 GOTO 20
6000 POKE V+21,0
6010 PRINT CHR$(147)
6020 FOR I=1 TO 10
6030 PRINT
6040 NEXT I
6050 PRINT TAB(13)"PUNTOS[1SPC]:";SC
6060 PRINT
6070 PRINT
6080 PRINT TAB(13);"OTRA[1SPC]?"
6090 GET X$
6100 IF X$="" THEN 6090
6110 IF X$<>"N" THEN RUN
6120 END
10000 GOSUB 30000
10010 FOR I=0 TO 29
10020 READ Q
10030 POKE 832+I,Q
10040 NEXT I
10050 FOR I=30 TO 62
10060 POKE 832+I,0
10070 NEXT I
10080 FOR I=0 TO 32
10090 READ Q
10100 POKE 896+I,Q
10110 NEXT I
10120 FOR I=33 TO 62
10130 POKE 896+I,0
10140 NEXT I
10150 FOR I=0 TO 62
10160 READ Q
10170 POKE 960+I,Q
10180 NEXT I
10190 FOR I=0 TO 62
10200 READ Q
10210 POKE 1024+I,Q
10220 NEXT I
10230 FOR I=0 TO 53
10240 POKE 1088+I,0
10250 NEXT I
10260 FOR I=18 TO 20
10270 POKE 1088+I*3,255
10280 POKE 1088+I*3+1,248
10290 POKE 1088+I*3+2,0
10300 NEXT I
10310 RETURN
20000 DATA 0,0,0,127,255,0,0,128,0
20005 DATA 1,192,7,3,240,7,31,252,15
20010 DATA 127,255,255,255,255,255
20020 DATA 255,255,255,127,255,254
20030 REM
20040 DATA 0,112,0,7,255,0,31,255,224
20050 DATA 127,255,248,255,255,254
20060 DATA 255,255,255,255,255,255
20070 DATA 127,255,254,31,225,248
20080 DATA 7,255,192,1,252,0
20090 REM
20100 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
20110 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
20120 DATA 13,128,0,5,0,0,5,0,0,7,0,0
20130 DATA 7,0,0,7,0,0,31,192,0,18,64,0
20140 DATA 23,64,0,7,0,0,0,0,0,0,0,0
20160 REM
20170 DATA 15,128,0,63,224,0,127,240,0
20180 DATA 127,240,0,255,248,0,255,248,0
20190 DATA 64,16,0,64,16,0,32,32,0
20200 DATA 32,32,0,23,64,0,23,64,0
20210 DATA 18,64,0,31,192,0,7,0,0,7,0,0
20220 DATA 7,0,0,5,0,0,5,0,0,13,128,0
30000 V=53248
30005 POKE V+21,0
30010 POKE V+11,230
30020 L=INT(RND(1)*100)+51
30030 POKE V+10,L
30040 RETURN
```

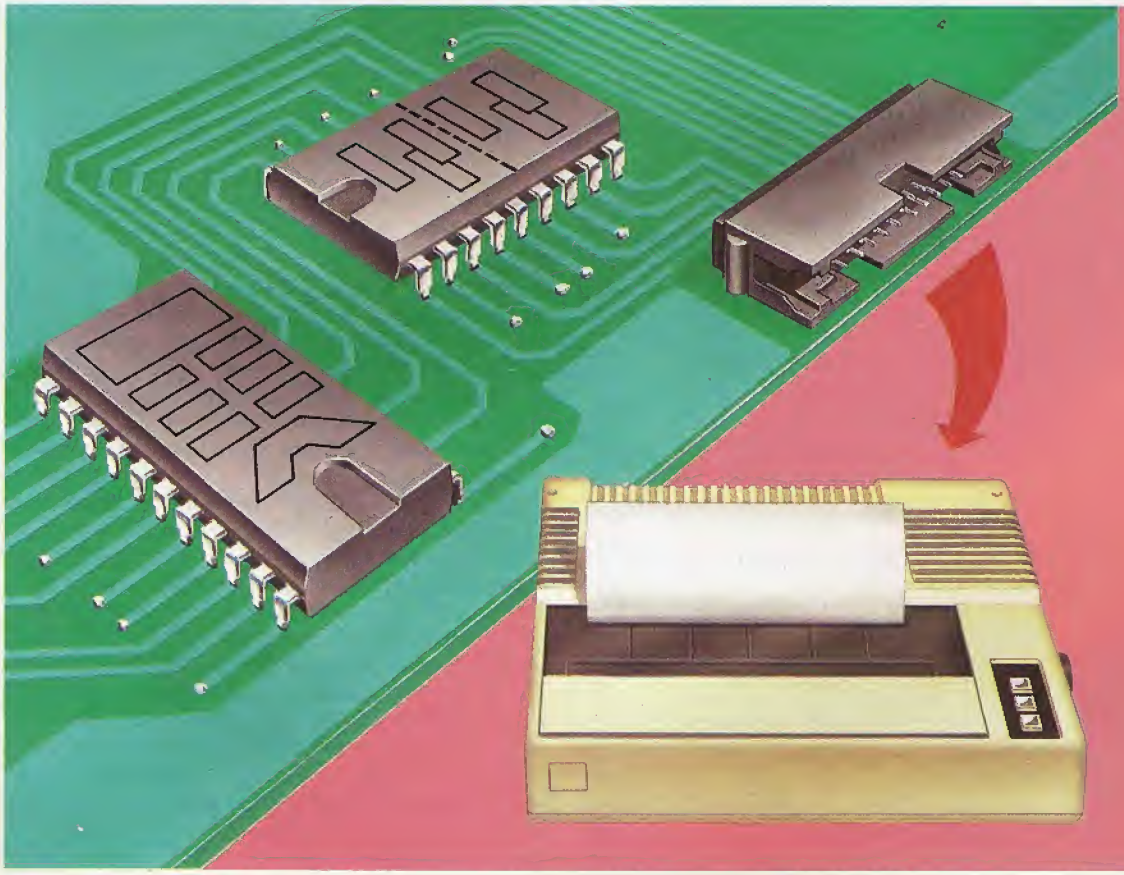
Puños fuera

Entre los aspectos más importantes de la programación en lenguaje assembly se halla el control de las entradas y salidas

El procesador 6809, al igual que el 6502 y a diferencia del Z80, no dispone de un espacio aparte de direcciones de E/S, ni tampoco de instrucciones particulares de E/S. En su lugar, los chips de la interface encargada del dispositivo de entrada/salida están asentados en la zona de direcciones normales y son manejados por medio de instrucciones de acceso a la memoria. Estos dispositivos son considerados por el procesador como otras tantas posiciones de memoria exactamente igual que las demás. La ventaja de este sistema es que resulta sencillo y rápido, y la desventaja es que se "roba" un bloque de direcciones a la memoria inutilizable en consecuencia para fines normales. Lo que significa que el 6809, aunque se dice que está provisto de un bus de direcciones de 16 bits y con 64 K de memoria de direccionamiento directo, en realidad sólo dispone de un máximo de 56 K sin hardware ni software de gestión de memoria.

Es posible para algunos dispositivos de E/S el estar conectados al bus de datos del sistema directamente, pero lo común es que haya un chip de interface entre ambos. Estos chips de interface son unos sutilísimos dispositivos tan complejos como el mismo microprocesador, y es corriente el uso de tales chips con un procesador de la misma familia, ya que facilita la tarea de su conexión y control. Los dos chips más comúnmente usados con el 6809 son el PIA 6802 o 6821 (*Peripheral Interface Adaptor*: adaptador de interface de periféricos), que trata las E/S en paralelo, y el ACIA 6850 (*Asynchronous Communications Interface Adaptor*: adaptador asíncrono de interface para comunicaciones), que trata las E/S en serie. Cada una tiene un número determinado de registros, y su control depende de la lectura y escritura de los contenidos almacenados en ellos, pues son considerados como otras tantas posiciones de memoria. Hay tres clases de registros:

- *Registros de control*: Son registros de sólo escritura; se almacenan en ellos valores programados en el chip según las opciones particulares que se desean, como, por ejemplo, el establecer una velocidad en baudios.
- *Registros de estado*: Son registros de sólo lectura, y sus valores indican el "estado" del chip. Por ejemplo, mostrarán si se ha recibido una entrada, o si se transmitió ya la última salida, o si se ha producido un error.



Periférico esencial

Las impresoras necesitan recibir los datos en formatos y velocidad determinadas: no es práctico que la CPU se encargue de tareas tan triviales, y por esta razón lo que hace ésta es enviar los datos de los caracteres al PIA, adaptador de la interface de periféricos, que emplea todo su tiempo en la comunicación con la impresora

Kevin Jones

• *Registros de datos*: Son registros que contienen datos de entrada o de salida, o sea, son de lectura-escritura o bien de lectura y escritura por separado. Para ahorrar espacio de memoria varios de estos registros ocupan la misma dirección. Por ejemplo, un registro de estado y otro de control pueden estar en la misma dirección; el que aparecerá en la dirección en cada momento dependerá de lo que se pretenda hacer, si leer o escribir en ella. De forma similar, un registro de datos de entrada y otro de salida pueden compartir la misma dirección.

El PIA 6820 contiene seis registros y ocupa cuatro bytes seguidos del espacio de memoria. El chip en sí está provisto de dos puertas independientes, cada una de las cuales utiliza tres registros. La sección de periféricos del chip tiene ocho líneas de datos y dos líneas de control por cada puerta. Las dos líneas de control se conectan en el periférico a las líneas de control adecuadas de tal modo que puedan servir para determinar el estado. La línea de control 1 sólo trata señales de control de entrada, pero la línea de control 2 puede ser programada para recibir o enviar señales de control.

Los tres registros son:

• Un registro de datos, que puede funcionar como entrada o salida, dado que cada bit puede ser activado independientemente.

• Un registro de dirección de datos, cuyos bits pueden servir individualmente para activar el bit correspondiente a entrada (0) o salida (1) en el registro de datos.

• Un registro mixto de control y estado.

El registro de dirección de datos y el registro de datos comparten la misma dirección. El estado de uno de los bits del registro de control indicará cuál de ellos está presente en un momento dado en la dirección. En la tabla al margen damos el desplazamiento de la dirección de base del chip, para obtener la dirección de cada uno de los registros.

Registro	Despl.
A de datos	0
A de dir. de datos	0
A de control/estado	1
B de datos	2
B de dir. de datos	2
B de control/estado	3

Los bits del registro de control/estado tienen la siguiente asignación:

Bit	Función
7	Bit de estado para la línea de control 1; se pone a uno cuando es recibida la señal de control y pasa automáticamente a cero cuando se lee el registro de datos
6	Bit de estado para la línea de control 2; funciona como el bit 7
5	Determina si la línea de control 2 se emplea para una entrada (0) o para una salida (1)
4	Determina la naturaleza de la señal de control en la línea 2
3	Si la línea de control 2 se establece para entrada, entonces un uno en este bit 3 permite la interrupción del bit 6; si era para salida, sirve para determinar la naturaleza de la señal
2	Selecciona el registro de datos (1) o el registro de dirección de datos (0)
1	Determina la naturaleza de la señal de control en la línea 1
0	Si está a 1 permite la interrupción del bit 7

De momento dejaremos de considerar el empleo de las interrupciones, así como el detalle de los efectos de los bits 1 y 4. Observe que cuando se escribe en el registro para establecer los bits de control es imposible afectar los bits 6 y 7.

Registro	Despl.
Reg. de control	0
Reg. de estado	0
Datos transmisión	1
Datos recepción	2

El primero de nuestros programas ejemplo crea y emplea un chip 6820 para controlar una impresora a través de una interface Centronics. El segundo especifica un gran número de líneas de control así como las ocho líneas de datos. No vamos a adentrarnos en su detalle, sólo haremos notar que una línea de control (llamada *strobe*) sirve para avisar a la impresora de que hay un carácter en camino. Ésta será conectada con la línea de control 2, la cual será puesta para salida. Otra señal de control (denominada *acknowledge*: reconocimiento) es usada por la impresora para indicar que está dispuesta para recibir el siguiente carácter. Ésta será conectada con la línea de control 1. Las ocho líneas de datos deben, naturalmente, ser conectadas a las ocho salidas de datos de la puerta PIA.

Para activar la puerta hay que seleccionar el registro de dirección de datos y establecer en salida la línea de control 2. Para emplear el chip debemos leer continuamente el registro de control/estado hasta que aparezca un 1 en el bit 7, que nos indicará que la impresora está preparada para recibir el carácter. Podemos a continuación escribir éste en el registro de datos, el cual envía automáticamente una señal de control por la línea de control 2. El bit 6 se pondrá a 1 cuando el carácter haya sido transmitido. Después se leerá el registro de datos para borrar los bits 6 y 7 y repetir el proceso hasta que haya sido transmitido el último carácter. El proceso de envío y recepción de señales de control entre el procesador y el periférico se conoce como *handshaking* (apretón de manos).

Supondremos que la dirección base del PIA se encuentra en una tabla de direcciones situada en $3000. Al entrar en la subrutina de impresión, el registro A del procesador contiene el índice de esta tabla, y el registro Y contiene la dirección de la serie de caracteres a imprimir. Esta cadena está almacenada en el formato normal, es decir, con el byte que indica su longitud en primer lugar. Hay dos subrutinas: una para activar la puerta y otra para imprimir la cadena.

El ACIA 6850 es un UART (transmisor/receptor asíncrono universal) empleado en la comunicación en serie, con el protocolo RS232 generalmente y un modem las más de las veces. Dispone de cuatro registros y ocupa dos direcciones. En el lugar de los periféricos se encuentran cinco conexiones con el chip: una línea para la transmisión de datos, otra para su recepción, y tres líneas de control para el *handshaking*, si así se necesita. Dos de ellas están destinadas a señales de control de entrada —la DCD (*Data Carrier Detect*: detección del portador de datos) y la CTS (*Clear To Send*: puesta a cero para envío)— y la última sirve para las señales de salida: RTS (*Request To Send*: petición de envío). El empleo de tales líneas se deduce de sus denominaciones, y han de conectarse a las líneas denominadas de igual modo dispuestas en el RS232 estándar.

Los cuatro registros del ACIA están escritos al margen de esta página. En el registro de control, el bit más significativo (el 7) sirve para permitir interrupciones en la recepción de datos. Los bits 5 y 6 se emplean para permitir o inhibir interrupciones en la transmisión y para determinar la naturaleza de la señal de control enviada por la línea RTS. Los bits 2, 3 y 4 sirven para determinar el tamaño del "paquete" transmitido en cada momento. Cuando se transmite un byte sobre un enlace en serie por lo general se envían al menos 10 bits, comenzando con el bit de inicio, detectado por el receptor para advertirle que hay datos en camino. Los datos en sí pueden ser siete u ocho bits, pero puede que se

añada un bit de paridad, es decir, un bit extra para la detección de errores en la transmisión. Finalmente, puede que se incluyan uno o dos bits de detección de final. He aquí las diferentes opciones:

Registro de control			Número de bits de datos	Paridad	Número de bits de final
Bit 4	Bit 3	Bit 2			
0	0	0	7	par	2
0	0	1	7	impar	2
0	1	0	7	par	1
0	1	1	7	impar	1
1	0	0	8	nula	2
1	0	1	8	nula	1
1	1	0	8	par	1
1	1	1	8	impar	1

Los dos bits menos significativos (el bit 0 y el 1) se emplean para determinar la velocidad de transmisión y de recepción. Esto se consigue estableciendo un divisor del paso de reloj. El 6850 no tiene reloj propio, por lo que debemos proporcionarle uno externo, comúnmente a 1 760 Hz.

Registro de control		Paso de reloj
Bit 1	Bit 0	divisor
0	0	1
0	1	16
1	0	64

En la tabla se omite la cuarta posibilidad (ambos bits a uno) que ocasiona la reinicialización total del chip. En el registro de estado los bits tienen las siguientes funciones:

Bit	Función
7	Petición de interrupción
6	Se pone a 1 si ha ocurrido algún error en la recepción
5	Se pone a 1 si hay una sobreescritura en el receptor, es decir, si los caracteres se sobreponen a los previamente recibidos
4	Se pone a 1 si ha habido un error de enmarque (*framing*) en la recepción, o sea, hay un número equivocado en los bits de inicio o de final
3	Se pone a 1 cuando se recibe una señal por la línea CTS
2	Se pone a 1 cuando se recibe una señal en la línea DCD
1	Se pone a 1 cuando el reg. de transm. de datos está vacío
0	Se pone a 1 cuando el reg. de datos recibidos está lleno

Nuestro programa del segundo ejemplo emplea un chip 6850 para recibir una cadena de caracteres, acabando en un retorno de carro, transmitidos de un terminal remoto. El principio consiste en programar el chip adecuadamente y hacer que un bucle compruebe si el registro de datos recibidos está completo. Cuando esto sucede, retiramos el byte de datos, lo cual reinicializa el bit 0 del registro de estado. Este proceso se repite hasta que el carácter recibido es el correspondiente al retorno de carro (el código 13 en ASCII). Nos despreocuparemos de cualquier posible error de transmisión, aunque no resulta nada difícil la comprobación a través de una máscara para los contenidos del registro de estado que compruebe si se ha puesto a 1 alguno de los bits indicadores de error de transmisión. Asumiremos un protocolo bastante común: 8 bits de datos, paridad nula, 2 bits de final y una velocidad de reloj dividida por 16. La primera subrutina programa el chip, la segunda recibe los datos.

Programa para PIA

```
TABLE   EQU   $3000          Subrutina para establ. Puerta A
        ORG   $1000
        ASLA                 Desplaza A a la izq. para
                             multiplicar por 2 (la tabla
                             contiene dirs. de 2 bytes)
        LDX   #TABLE
        LDX   A,X            Obtiene la dir. base del PIA
        CLR   1,X            Obtiene acceso al registro de
                             dirección de datos
        LDB   #%11111111
        STB   ,X             Pone a 1 los bits para salida
        LDB   #%00101100     Inhibe las interrupciones,
        STB   1,X            activa lín. cont. 2 para sal. y
                             selec. el reg. de datos
        RTS
                             Subrutina para imprimir la cadena
                             cuya dirección se encuentra en Y
        ASLA                 Desp. A a la izq. para mult. por 2
        LDX   #TABLE         Obtiene dir. base del PIA
        LDX   A,X
        LDA   ,Y+            Obtiene long. de la cad. en A
LOOP1   BEQ   FINISH         Comprueba si long. es cero
LOOP2   LDB   1,X            Compr. si está disp. para sig. bit
        ANDB  #%10000000     Enmascara bits excepto 7
        BEQ   LOOP2          Si no está preparada
        LDB   ,Y+            Obtiene el siguiente carácter
        STB   ,X             Lo imprime
LOOP3   LDB   1,X            Comprueba si fue transmitido
        ANDB  #%01000000     Mira el bit 6
        BEQ   LOOP3          Bucle si no está aún prepar.
        LDB   ,X             Lee reg. de datos para poner a
        DECA                 0 los bits de éstado
        BRA   LOOP1          Resta 1 a la longitud
FINISH  RTS                  Toma el sig. carácter
```

Programa para ACIA

```
TABLE   EQU   $3000          Subrutina para programar el 6850
        ORG   $1000
                             Subrutina para est. el ACIA
ACIAST  ASLA                 Desp. A a la izq. para mult.
                             por 2 (tabla dirs. de 2 bytes)
        LDX   #TABLE         Obtiene dir. base del ACIA
        LDX   A,X
        LDA   #%00000011     Reinicialización total del ACIA
        STA   ,X             Pone en reg. de control
        LDA   #%00010001     Progr. el ACIA (8 bits de datos,
        STA   ,X             no hay paridad, 2 bits de final)
        RTS
                             Subrutina para aceptar cadena de
                             caracteres
BUFFER  EQU   $4000          Algún lugar para colocar cad.
CR      EQU   13             Código ASCII retor. de carro
        BSR   ACIAST         Establece el ACIA. El registro
        LDY   #BUFFER        X contiene la dir. del ACIA
                             Destino en Y
LOOP    LDB   ,X             Obtiene el estado
        ASLB                 Desp. el bit 7 fuera del reg. B
        BCC   LOOP           y lo introd. en flag de arras. CRC
        LDA   1,X            Retrocede si ese bit no se
        STA   ,Y+            puso a 1, es decir, que no se
        CMPA  #CR            solicitó aún interr. alguna
        BNE   LOOP           Toma el byte de datos
        RTS                  Lo almacena
                             ¿Se trata del retor. de carro?
                             Siguiente carácter
```

Guerra en la galaxia

Encuentro con los Zylones

Mapa galáctico

Ian McKinnell

"Star raiders", de Atari, es un desarrollo, especial para ordenadores personales, del popular juego "Star trek"

En *Star raiders*, el jugador asume el papel de comandante de la nave espacial Star Raider y viaja a través de la galaxia en busca de las naves de los enemigos, los Zylones. El juego requiere el empleo del teclado además del control por palanca de mando. Después de desplazarse hasta un sector de la galaxia, pulsando la tecla F se visualiza la panorámica frontal desde la cabina. La posición de las naves Zylon se señala mediante indicadores en la parte inferior de la pantalla, y pulsando la tecla L el jugador llama al "explorador de largo alcance", que proporciona una panorámica del sector de la cuadrícula en curso con la nave del jugador en el centro y las Zylon formadas en la distancia.

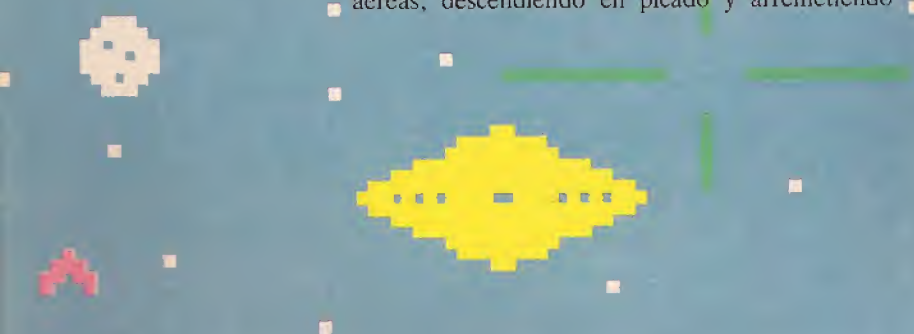
El jugador debe entonces atacar al enemigo, ya sea utilizando los motores normales, lo que bien podría conducir a que las naves Zylon escaparan, o bien adentrándose en el "hiperespacio" (lo que se consigue pulsando H), en cuyo caso la distancia se cubrirá en unos pocos segundos. Antes de lanzarse al hiperespacio, debe utilizarse el ordenador perseguidor pulsando T; si no se hace esto, el salto al hiperespacio bien podría conducir a que la nave del jugador acabara en un sector totalmente desconocido de la galaxia. Otros factores a considerar son la utilización del ordenador de ataque (al que se accede mediante la tecla C) y los escudos defensores del Star Raider, que se encienden mediante el empleo de la tecla S.

Una vez abandonado el hiperespacio, el ordenador enciende una luz intermitente de "alerta roja" y comienza la batalla. Las naves Zylon atacan la Star Raider desde todos los lados, haciéndose cada vez más grandes a medida que se acercan. El control con la palanca de mando permite al jugador hacer rotar la nave en todas las direcciones, y la velocidad de la misma se determina mediante el empleo de las teclas numéricas. La medida de control ofrecida permite que el jugador intervenga en escaramuzas aéreas, descendiendo en picado y arremetiendo contra el enemigo. Pero éstas consumen muchísima energía y la flota enemiga hará blanco repetidamente en la Star Raider, lo que obligará al usuario a visitar la base estelar para reabastecerse de combustible y efectuar las necesarias reparaciones. Deberá, entonces, trasladarse a una cuadrícula que contiene una estrella; a medida que la Star Raider se acerca, la estrella irá creciendo y se transformará en un enorme platillo volante amarillo. Entonces se produce la maniobra más difícil del juego, con el jugador intentando colocarse en órbita alrededor de la base. En primer lugar, es preciso que la nave esté fija en los visores de la nave, y el jugador debe retardar el impulso de ésta hasta que el indicador de distancia al objetivo marque cero. Una vez sucede esto, se puede detener a la Star Raider y, si la maniobra ha sido correcta, el mensaje ORBIT ESTABLISHED (establecida órbita) aparecerá intermitente en la parte superior de la pantalla. Durante esta aproximación se debe tener un enorme cuidado, puesto que es extremadamente fácil errar el objetivo. Después de establecida la órbita, de la base estelar emerge una nave auxiliar de reaprovisionamiento de combustible y se acopla con la Star Raider, permitiendo que la nave retorne al combate.

De cuando en cuando aparece en la pantalla un mensaje STARBASE SURROUNDED (base estelar rodeada) y el jugador debe, entonces, dirigirse apresuradamente hacia la estación amenazada para impedir su destrucción. Cuando intenta defender la base estelar, el jugador debe dedicar sumo cuidado a evitar que las armas de la Star Raider hagan blanco en la base estelar.

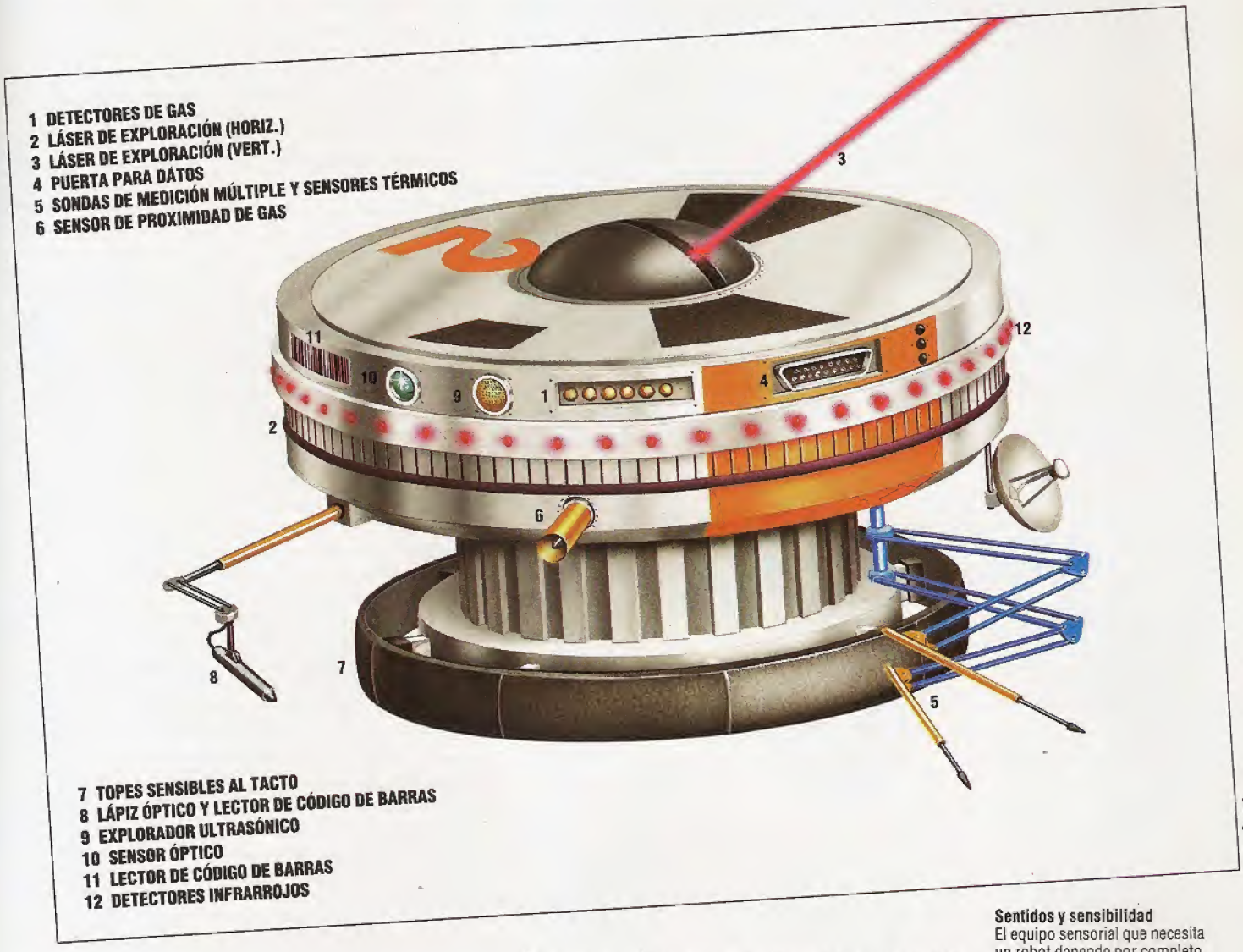
Star raiders ofrece cuatro niveles de juego, que van desde "novato" a "comandante". En los niveles inferiores, el jugador no necesita preocuparse demasiado por el daño que se le pueda infligir a la Star Raider, dado que hay muy pocos Zylones y la puntería de éstos no es muy certera. En los niveles superiores resulta sumamente difícil sobrevivir.

Star raiders: Para todos los ordenadores Atari
Editado y distribuido por: Atari Co., AUDELEC, Compás de la Victoria, 3. 29012 Málaga, España
Autores: Atari
Palanca de mando: Necesaria
Formato: Cartucho



Liz Dixon

El mundo exterior



Steve Gross

Sentidos y sensibilidad
El equipo sensorial que necesita un robot depende por completo de sus funciones; pero cuanto más generales sean los usos a que se destine el robot, más probable es que necesite más sensores. El robot de la ilustración ofrece ejemplos de la mayoría de los sensores posibles existentes, si bien no es probable que ningún robot pueda, él solo, incorporar una gama tan amplia

En esta ocasión estudiaremos de qué manera pueden los robots percibir lo que sucede a su alrededor

El sistema sensorial del hombre es algo que damos por sentado, pero una persona que careciera de todos los sentidos estaría totalmente desvalida. Sin el sentido de la vista, se tropezaría con los objetos cuando se intentara caminar; sin el tacto, uno ni siquiera se enteraría de que había tropezado; la sordera total significaría que uno ni siquiera podría recibir una advertencia de que estaba a punto de llevarse un objeto por delante. De hecho, ni siquiera se podría caminar, porque para informarle al cerebro de la forma en que el cuerpo se está moviendo son necesarios los sentidos internos.

Hemos explicado cómo se puede desplazar un robot, pero debemos asimismo proporcionarle, antes de que pueda actuar con independencia, un sistema sensorial. Puede resultar apasionante tratar de diseñar un robot que posea *todos* los sentidos humanos: entonces podría percibir el mundo de forma muy similar a como lo percibimos nosotros. Por el momento, sin embargo, esto es imposible. Los temas relativos a la comprensión de lo que se ve y se habla son tan complejos que nos ocuparemos de ellos con profundidad en un futuro capítulo. Aquí nos concentraremos en sencillas aproximaciones a la vista y el oído que están muy por debajo del nivel de complejidad que poseen los humanos.

Es bastante sencillo hacer que un robot "vea" las cosas proporcionándole un sensor luminoso (por lo general, una célula fotoeléctrica) que produce un voltaje que varía con la cantidad de luz que recibe. Éste es un sensor de visión muy rudimentario, pero con él se pueden conseguir buenos efectos. Por ejemplo, se puede hacer que un robot se "dirija" hacia una luz brillante de modo muy similar a como se le hace seguir una línea (véase p. 1121). Esto se

puede utilizar para permitir que el robot localice un punto de energía para recargarse cuando se le agoten las pilas. (Tenga en cuenta que ello requerirá que el robot posea un sensor interno para controlar el estado de éstas: así "sabrá" cuándo están bajas.)

Esta sencilla célula fotoeléctrica puede permitir que un robot lleve a cabo numerosas tareas. Un robot instalado en una cadena de montaje podría estar capacitado para verificar si un componente está presente detectando la diferencia de brillo debida a la ausencia de aquél; esta tarea se puede simplificar disponiendo que la iluminación sea la adecuada para que se acentúe tal cambio. El robot podría detectar variaciones de color si se incorporaran tres células fotoeléctricas, respondiendo cada una de ellas a una luz de distinto color; el rojo, el verde y el azul cubrirían el espectro visible. Un robot de estas características se podría programar de modo que captara los ladrillos rojos de una pila que contuviera ladrillos de muchos colores diferentes. Esto produce la impresión, a partir de un sensor muy simple, de un comportamiento "inteligente".

Si al robot se le proporciona un micrófono, podrá "oír" señales acústicas. No "entenderá" lo que está oyendo, pero esto no tiene por qué ser importante: repitiendo varias veces un conjunto de instrucciones, el robot puede construir un "modelo" de sonido para cada una de ellas, que le permitirá comparar instrucciones nuevas con las oídas previamente. El número de instrucciones a las cuales podrá responder será limitado, pero podremos decirle que vaya "hacia adelante", etc., y obedecerá.

Un robot puede, asimismo, poseer un sencillo sentido del tacto. En su diseño se pueden incorporar microinterruptores de modo tal que establezcan una conexión eléctrica siempre que se les aplique una presión. Éstos carecen de la complejidad del sentido humano del tacto, pero aun así pueden ser muy útiles. Por ejemplo, sensores táctiles montados alrededor del borde de un robot móvil pueden permitirle responder inteligentemente a cualquier obstáculo: el robot será capaz de volverse hacia atrás ante la obstrucción y probar otra ruta. Sensores táctiles incorporados en una mano permitirán que éste "sepa" cuándo tiene algo a su alcance de modo que pueda responder en consecuencia.

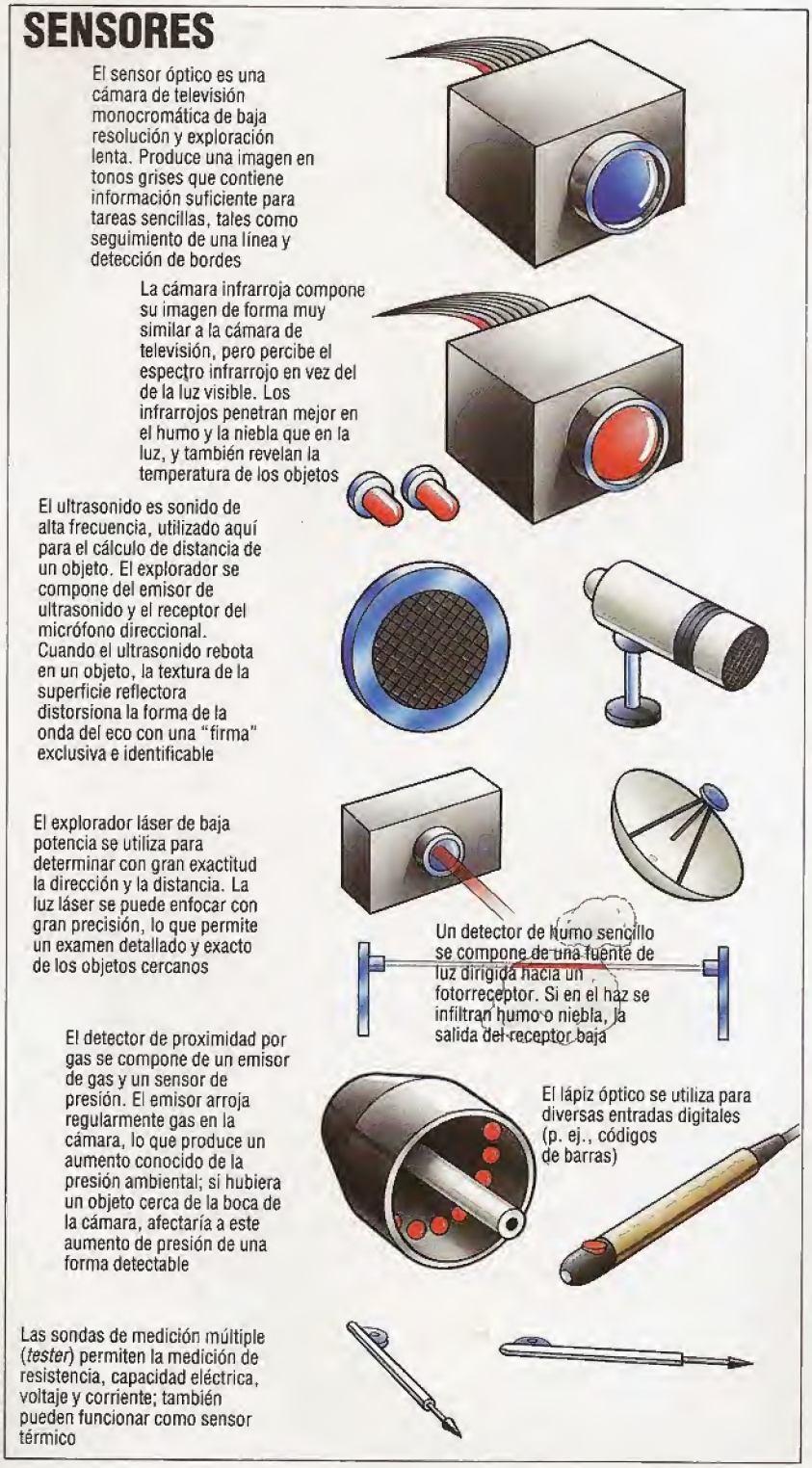
Se pueden utilizar detectores de humo o gas para darle al robot una especie de sentido de olfato. Los detectores de gas suelen utilizar un elemento sensorial (tal como un cable de platino) que responde ante la presencia de ciertos gases, alterando, por consiguiente, la corriente eléctrica que fluye a través del elemento. Los detectores de humo poseen dos cámaras: una cerrada, que actúa a modo de referencia o "control", y la otra abierta. Ambas cámaras contienen helio ionizado y la cantidad de partículas cargadas de la cámara abierta varía cuando hay humo. Un detector que cuente las partículas cargadas que hay en cada cámara registrará una diferencia entre las dos cuando haya humo.

Por el momento, parecería no haber forma alguna de dotar a un robot de sentido del gusto. No obstante, aplicando los métodos que hemos sugerido, al menos tendremos un robot que puede ver, oír, sentir y oler lo suficientemente bien como para detectar un incendio en el edificio, correr hacia las llamas, evitando los obstáculos que encontrará en su camino, y, en caso de que tuviera un extintor de incendios en su efector final ("mano"), sofocar el fuego con espuma.

Ganancia potencial

Pero limitando un robot al tipo de sentidos que poseemos los humanos estamos perdiendo mucho de su potencial. No existe ninguna razón por la cual se haya de restringir al robot a detectar cosas de la forma en que las detectamos nosotros. Un enfoque mejor sería el de considerar qué sentidos se le pueden proporcionar al robot y decidir si los mismos tienen algún uso práctico.

SENSORES



Kevin Jones

Un buen ejemplo de esto nos lo dan los brazos-robot. Vamos a suponer que deseamos que un robot recoja un objeto de un lugar y después lo coloque en algún otro sitio. Una forma de hacer esto es fijar topes alrededor del brazo, de modo que éste sólo pueda recorrer una distancia máxima preestablecida en cualquier dirección dada. El brazo se balancearía hasta llegar a los topes, en cuyo punto (si todo estuviera situado correctamente) la mano estaría directamente encima del objeto a coger. Después de asir el objeto, el brazo se balancearía en la dirección contraria hasta que otro tope le haría saber que debe soltarlo. Éste es un ejemplo sencillo y su uso se está reduciendo cada vez más, pero demuestra que a los robots se los puede dotar de sentidos de los cuales carecemos.

Tal vez sea más ilustrativo el ejemplo del empleo de la "vista" por parte de un robot. Los seres humanos sólo perciben la luz visible (una gran parte del espectro electromagnético es invisible al ojo humano), pero no hay ninguna razón por que el robot deba sufrir esta restricción. En vez de células fotoeléctricas se podrían instalar detectores infrarrojos; éstos permitirán medir la cantidad de calor generada por un objeto. Los robots industriales pueden hacer uso de estos detectores para alejarse de cualquier objeto cuya temperatura sea, por ejemplo, peligrosamente alta. Pero un robot también podría detectar la tibieza de un cuerpo humano, ¡de modo que usted podría programar su robot personal para que saliera corriendo a su encuentro cuando atravesara la puerta de su casa! También se puede hacer que los robots detecten campos magnéticos. Esto ya se ha analizado en relación a los robots que siguen una huella trazada en el suelo, pero esta facilidad también sería útil para aplicaciones en las cuales un robot hubiera de diferenciar entre materiales magnéticos y no magnéticos.

Los sensores de proximidad no tienen un equivalente exacto en el hombre; son sólo dispositivos que pueden detectar la cercanía de un objeto. Con este fin, los seres humanos se sirven de una combinación de vista y tacto, pero un sencillo sensor de proximidad es igualmente apto para el empleo en robots. Tales sensores trabajan de diversas maneras. Un tipo utiliza un chorro de aire arrojado a través de una boquilla; cualquier objeto que haya en el recorrido del chorro desviará el aire nuevamente hacia la boquilla. Esto crea una presión de retroceso que puede ser detectada por un transductor de presión, advirtiendo, por tanto, al robot de que hay algo cerca. Otro tipo se basa en el hecho de que un circuito eléctrico con un condensador cambiará su comportamiento si se está acercando a otro objeto. Una "fuga" entre el condensador y el objeto (que tendrá una capacidad eléctrica propia) le informará al robot que hay otro cuerpo próximo a él.

Transductores

Existen, asimismo, detectores ultrasónicos que funcionan emitiendo una señal ultrasónica y captando luego el eco producido por el objeto cercano. El tiempo transcurrido entre la señal y el eco proporciona una medida exacta de la distancia a la que se halla el objeto. Este método es similar al que utilizan los murciélagos para conocer su situación, y el principio también se emplea en algunas cámaras de enfoque automático.

Sin sentidos no hay sensación Este brazo-robot industrial está sacando las piezas de hierro fundido de los moldes cuando aún están demasiado calientes para que las manos humanas las puedan tocar. El robot, por supuesto, es insensible al calor y, en consecuencia, realiza el trabajo más rápidamente

Los sensores láser son aún más sofisticados. Éstos dirigen un haz sobre un objeto, que entonces refleja la luz del láser de vuelta hacia el sensor. Mediante la comparación de ambos haces se puede determinar la distancia del objeto con una precisión asombrosa. Esta técnica se puede utilizar para grandes distancias. Durante el primer alunizaje de una nave tripulada se colocó en el satélite un reflector para permitir que un sensor láser pudiera medir la distancia exacta entre la Tierra y la Luna. Se afirma que el margen de error de esta forma de medición ¡es de 15 cm para una distancia de 384 400 km!

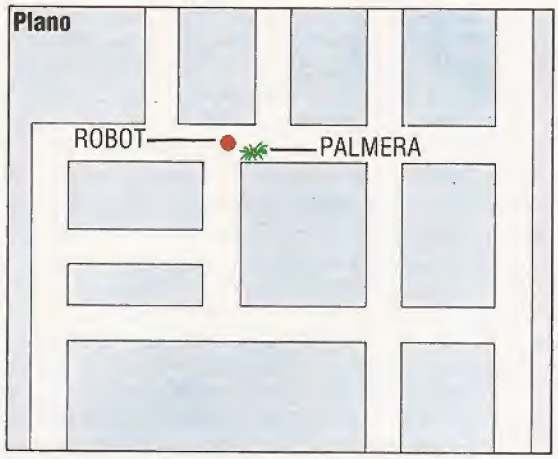
Los sensores de fuerza son un medio para obtener información táctil mediante medios más sofisticados que los microinterruptores mecánicos. Éstos operan midiendo el cambio producido en las propiedades eléctricas de un cristal piezoeléctrico cuando éste es sometido a presión, o calculando el cambio en la conductividad de gránulos de grafito de carbón bajo presión (utilizando una técnica idéntica a la empleada en el micrófono de carbón). Alternativamente, se pueden utilizar indicadores de tensión para medir fuerzas grandes detectando los cambios producidos en la resistencia eléctrica de un cable mientras el mismo es estirado.

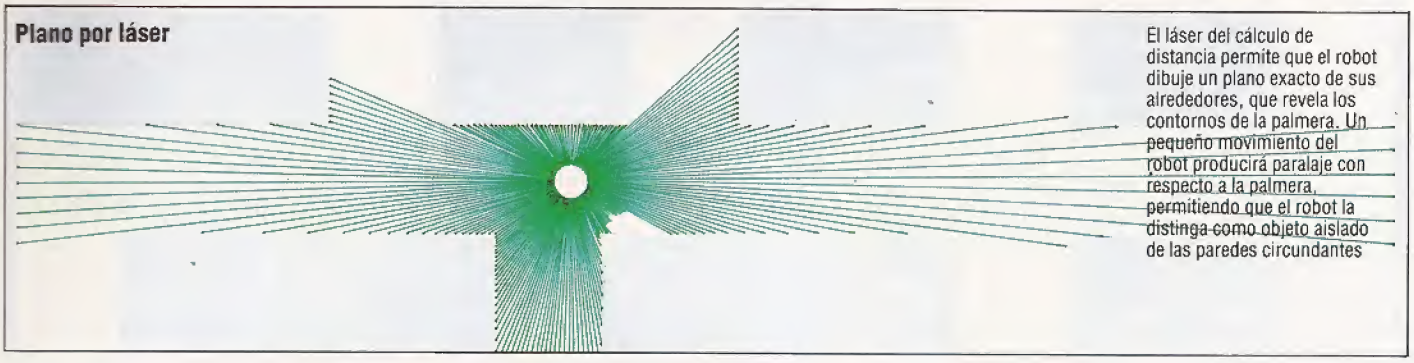
Estos sensores de robots se agrupan bajo la denominación común de *transductores*, dado que toman una medición de una forma (que puede ser luz, sonido o presión) y la convierten ("transducen") a otra que de alguna manera representa la medición original. En un robot controlado por ordenador, los transductores casi invariablemente convierten la medición en una señal eléctrica que puede ser binaria (es decir, la señal eléctrica está presente o no) o analógica (la señal varía en la medida en que la medición original cambia). En este último caso, la señal eléctrica se debe convertir a una forma que pueda entender el ordenador mediante el empleo de un convertidor A/D.

Es justo reconocer que los sentidos de un robot no son ni tan amplios ni tan eficaces como sus equivalentes humanos. Pero el robot posee más sentidos, y éstos se están perfeccionando día a día.

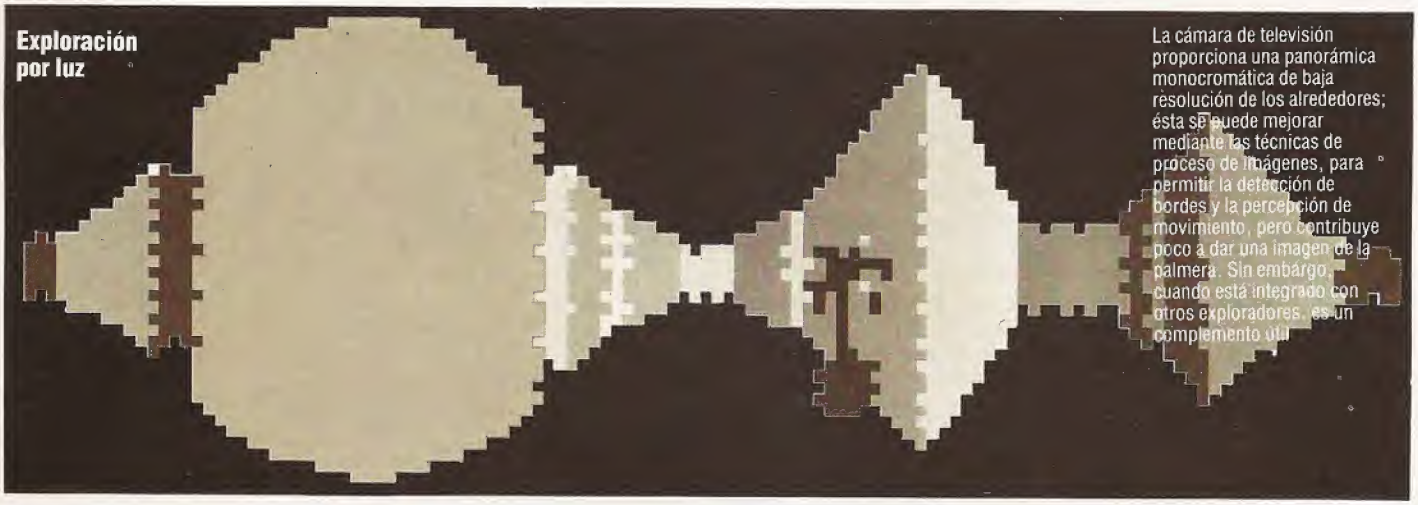
Sentidos mediante sensores

La percepción del mundo exterior es el mayor problema del robot, y aún lo es más cuanto mayor sea la gama y la complejidad de su equipo sensorial. No hay ningún sensor individual que proporcione una imagen completamente informativa, y algunos incluso parecen contradecirse. El nivel al cual el robot pueda integrar y comparar las entradas de sus varios sensores es lo que proporciona la medida de su "conciencia" externa.
En este ejemplo, el plano muestra que el robot se halla en un corredor cuyas paredes están pintadas de blanco; sólo hay una fuente de luz, de modo que la iluminación de una pared depende de su orientación. Cerca del robot hay una palmera

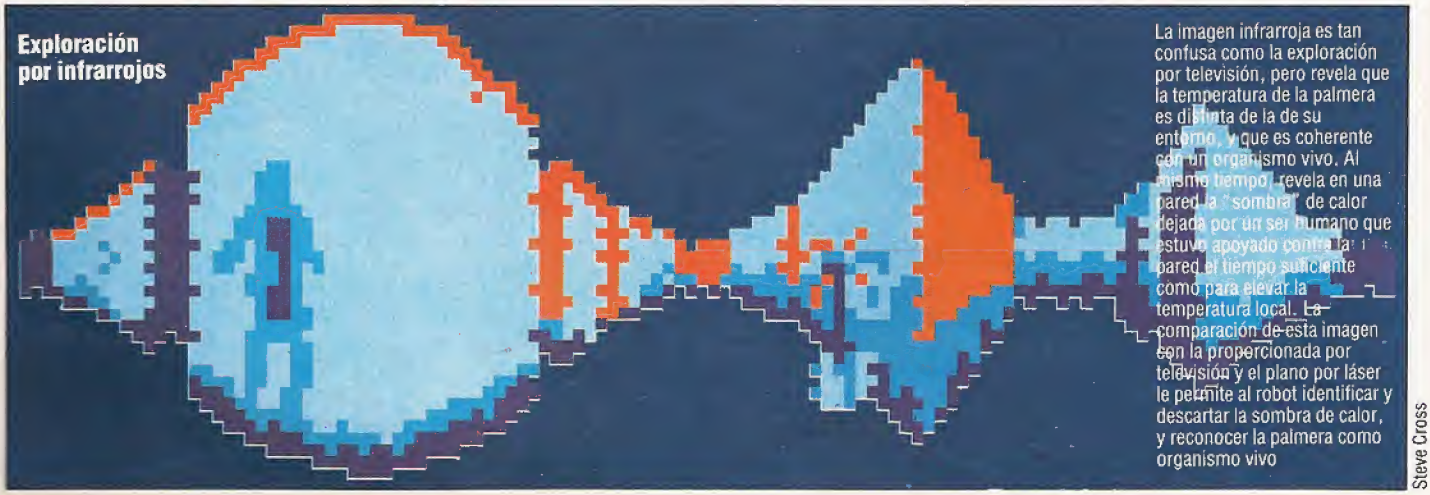




El láser del cálculo de distancia permite que el robot dibuje un plano exacto de sus alrededores, que revela los contornos de la palmera. Un pequeño movimiento del robot producirá paralaje con respecto a la palmera, permitiendo que el robot la distinga como objeto aislado de las paredes circundantes



La cámara de televisión proporciona una panorámica monocromática de baja resolución de los alrededores; ésta se puede mejorar mediante las técnicas de proceso de imágenes, para permitir la detección de bordes y la percepción de movimiento, pero contribuye poco a dar una imagen de la palmera. Sin embargo, cuando está integrado con otros exploradores, es un complemento útil



La imagen infrarroja es tan confusa como la exploración por televisión, pero revela que la temperatura de la palmera es distinta de la de su entorno, y que es coherente con un organismo vivo. Al mismo tiempo, revela en una pared la "sombra" de calor dejada por un ser humano que estuvo apoyado contra la pared el tiempo suficiente como para elevar la temperatura local. La comparación de esta imagen con la proporcionada por televisión y el plano por láser le permite al robot identificar y descartar la sombra de calor, y reconocer la palmera como organismo vivo

Steve Cross

Dígitos visualizados

En este capítulo agregamos dos visualizaciones de siete segmentos a nuestro sistema de la puerta para el usuario

Para visualizar dígitos hexadecimales se requieren cuatro bits (cuatro bits nos dan 16 permutaciones de ceros y unos). En consecuencia, cualquier número de ocho bits se puede representar utilizando dos dígitos hexadecimales: uno para los cuatro bits inferiores y otro para los cuatro superiores. A pesar de que cada visualización consta de siete segmentos LED, las diversas combinaciones de segmentos se pueden "activar" mediante cuatro líneas de entrada si se incorpora un circuito lógico de decodificación.

Los decodificadores son circuitos que traducen instrucciones del ordenador a sus periféricos en señales eléctricas, y viceversa. En nuestra serie sobre lógica construiremos nuestro propio circuito decodificador (véase p. 626), pero para este ejercicio podemos comprar un circuito lógico ya hecho. Éste es el chip 7447 de la lista de componentes.

El decodificador para cada visualización acepta cuatro líneas de entrada provenientes de la puerta para el usuario y, a través de una secuencia de puertas lógicas, proporciona siete salidas. El circuito lógico se ha diseñado de modo tal que si, pongamos por caso, las cuatro líneas de entrada fueran 0111, entonces se encenderían las barras correspondientes para visualizar el número 7 (0111 en binario equivale a 7 en hexadecimal). En la página 1167 podemos ver la tabla de verdad para ello.

Los dígitos hexadecimales mayores que nueve generalmente se representan mediante las siete primeras letras del alfabeto: de la A a la F. Se observará que el chip decodificador que estamos utilizando posee patrones un tanto extraños para representar estos dígitos. Es probable que estos patrones se puedan generar utilizando los circuitos lógicos requeridos para los dígitos del cero al nueve. Sería necesaria más lógica decodificadora para visualizar los seis últimos dígitos hexas de la forma alfabética más habitual, de modo que descartando la lógica extra y empleando símbolos distintos para estos dígitos se reduce la cantidad de puertas lógicas del decodificador, rebajando, por consiguiente, el costo que supone la fabricación del chip.

Una vez construido el circuito de visualización podemos visualizar continuamente el contenido del registro de datos de la puerta para el usuario en hexa, empleando las ocho líneas de entrada proporcionadas. Hay suficientes líneas disponibles para utilizar las dos visualizaciones simultáneamente, pero esto no sucede así en muchas aplicaciones, y varias visualizaciones de siete segmentos deben compartir las mismas líneas de datos. Para que cada visualización pueda mostrar distinta información al mismo tiempo, se utiliza una técnica llamada *multiplexión*. En esencia, las líneas de datos del decodificador de visualización se pasan rápidamente de una visualización a la siguiente, cambiando también de la forma correspondiente los datos presentes en las líneas. Si esto se realiza con rapidez suficiente, todas las visualizaciones multiplexadas de esta manera parecerán parpadear continuamente, visualizando cada una los datos que estén presentes en el instante en que se conecta a las líneas de datos.

Lista de componentes

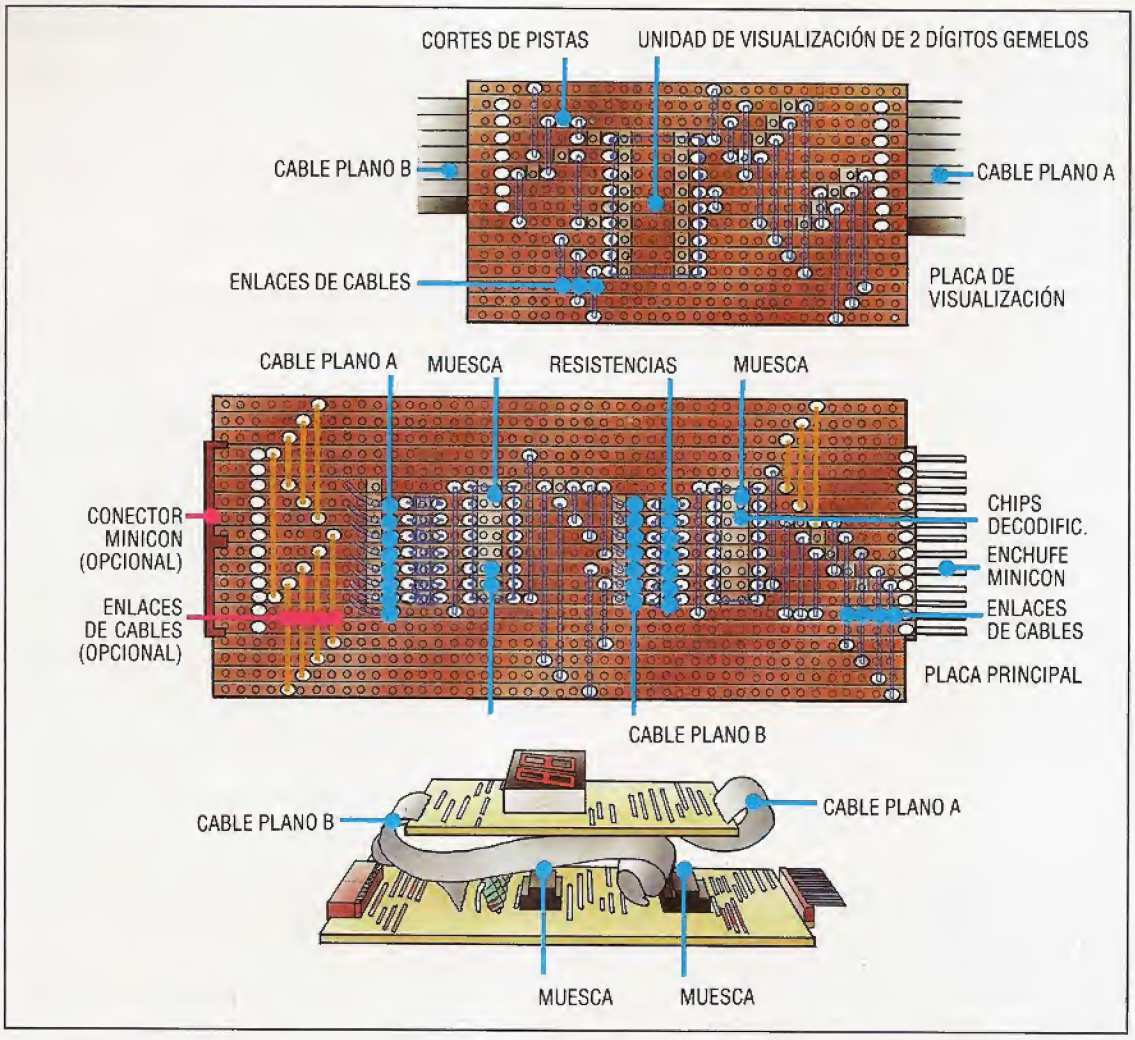
Cantidad	Artículo
14	Resistencia 330 ohmios 0,4 vatios
2	BCD 7447 a decod. 7 segmentos
1	Visual. de 2 dígitos gemelos de ánodo común
2	Conector de chip DIL de 16 patillas
1	Con. minicon ángulo recto 12 vías
1	Ench. ángulo recto minicon 10 vías
	Cable plano de 8 vías*
	Cable plano de 7 vías*
	Cable pelado estañado*
1	Veroboard de 50 agujeros×36 franjas
1	Caja plástica de 116×61×36 mm

* Estos componentes pueden haberle sobrado de proyectos anteriores. El conector de 12 vías sólo es necesario si desea ampliar el bus del sistema

Podemos demostrar el principio de la multiplexión utilizando las dos visualizaciones de siete segmentos que estamos construyendo. Puesto que el decodificador de visualización representa al decimal 15 con un vacío, podemos emplear este número para borrar una visualización mientras se ilumina la otra. El siguiente programa, el ejecutarse, solicita un dígito a visualizar y después va mostrando el dígito en ambas visualizaciones al mismo tiempo. Sin embargo, se incluye una rutina que inserta una demora para retardar la oscilación entre las dos visualizaciones. La demora se inserta mientras se pulsa la barra espaciadora. Podemos ver, al ejecutar el programa y pulsar la barra, que en realidad el dígito salta una y otra vez de una visualización a la otra. Cuando se vuelve a liberar la barra espaciadora, se elimina la demora y el salto es más rápido.

```
 10 REM MULTIPLEXION BBC
 20 RDD=&FE62:REGDAT=&FE60
 30 ?RDD=255
 40 izquierda_vacio=15*16
 50 derecha_vacio=15
 55 :
 60 REPEAT
 70 INPUT"DATOS A MULTIPLEXAR";datos
 80 ?REGDAT=datos+izquierda_vacio
 90 PROCmas_despacio
100 ?REGDAT=datos*16+derecha_vacio
110 PROCmas_despacio
120 GOTO80
130 END
140 :
150 DEF PROCmas_despacio
155 REM SE ESTA PULSANDO LA BARRA ESPACIADORA?
160 IF INKEY(-99)=-1 THEN PROCdemora
170 ENDPROC
```

Planificándolo todo
Corte la veroboard en los dos tamaños necesarios (19 pistas de 46 agujeros; 15 pistas de 28 agujeros). Haga primero los cortes de pistas en ambas placas. Suelde los dos conectores de chip en su sitio, luego los enlaces de cables y las resistencias. Si no se requiere el conector para ampliación del bus, entonces omita los enlaces que en la ilustración aparecen en rojo. Coloque el enchufe minicon y el conector (opcional) en la placa principal, y suelde la unidad de visualización en su sitio (los puntos hacia el extremo del conector del tablero). Suelde los cables planos de conexión, de modo que vayan directamente de placa a placa sin retorcerse. Ahora enchufe los chips; asegúrese de que estén orientados tal como indica la ilustración




```
180 :
190 DEF PROCdemora
200 FOR I=1 TO 500:NEXT
210 ENDPROC
```

```
 10 REM MULTIPLEXION CBM 64
 30 RDD=56579:REGDAT=56577
 40 POKERDD,255
 50 IV=15*16:DV=15
 60 INPUT"DATOS A MULTIPLEXAR";DT
 70 POKEREGDAT,DT+IV
 80 GOSUB1000:REM MAS DESPACIO
 90 POKEREGDAT,DT*16+DV
100 GOSUB1000:REM MAS DESPACIO
110 GOTO70
120 :
1000 REM S/R MAS DESPACIO
1010 GET$
1020 IFA$=" "THENGOSUB2000:REM DEMORA
1030 RETURN
1999 :
2000 REM S/R DEMORA
2010 FORI=1 TO 250:NEXT
2020 RETURN
```

Una aplicación sencilla es utilizar las visualizaciones de siete segmentos gemelos como un contador hexadecimal. Esta visualiza una cuenta del número de impulsos entrados en la puerta para el usuario desde un simple interruptor conectado a una línea de la puerta. A primera vista esta tarea parece trivial, hasta que uno comprende que se requieren las ocho líneas de la puerta para el usuario para la visualización, sin que quede ninguna para entrada. Si especificamos una de las líneas para entrada, pongamos por caso la línea 0, entonces el sistema de E/S del ordenador siempre retendrá esta línea alta, independientemente de qué número esté presente en el registro de datos. Si en éste fuera a colocarse 128 (10000000 en binario), entonces éste instantáneamente cambiaría a 129 (10000001) porque la línea 0 estaría retenida alta para entrada. Esto, como es obvio, daría valores de contador incorrectos en las visualizaciones. La solución estriba en aplicar una técnica similar a la multiplexión. Si utilizamos la línea 0 para que acepte entrada sólo durante un breve período y utilizamos todas las líneas para salida durante un período más largo, entonces en las visualizaciones parecerá brillar continuamente el valor correcto del contador, con apenas una oscilación del valor incorrecto producido por establecer momentáneamente la línea 0 para entrada.

```
 10 REM CONTADOR BBC
 20 RDD=&FE62:REGDAT=&FE60
 50 contador=0
 55 :
 60 REPEAT
 70 PROCinput
 72 PROCsumar
 73 FORI=1TO40
 75 PROCvisualizacion
 77 NEXT I
 80 UNTIL contador>255
 90 END
999 :
1000 DEF PROCsumar
```

```
1010 IF bandera=1 THEN contador=contador+1
1050 ENDPROC
1499 :
1500 DEF PROCinput
1510 ?RDD=254
1515 bandera=0
1520 IF(?REGDAT AND 1)=0 THEN bandera=1
1525 REPEAT UNTIL (?REGDAT AND 1)=1
1530 ENDPROC
1999 :
2000 DEF PROCvisualizacion
2010 ?RDD=255
2030 ?REGDAT=contador
2040 ENDPROC
```

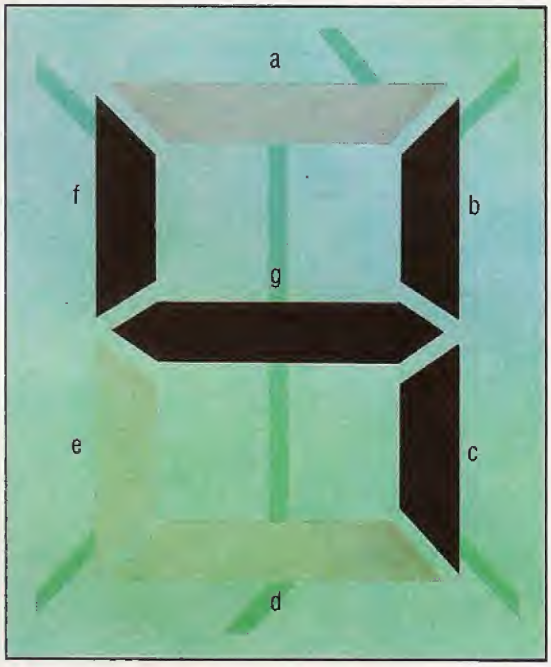
```
  10 REM CONTADOR CBM 64
  30 RDD=56579:REGDAT=56577
  40 CC=0:REM INIC CONTADOR
  50 :
  60 GOSUB1000:REM ENTRADA
  70 GOSUB2000:REM SUMAR
  80 FOR I=1TO20
  90 GOSUB3000:REM VISUALIZACION
 100 NEXT I
 110 IF CC<255 THEN60
 120 END
 999 :
1000 REM S/R ENTRADA
1010 POKERDD,254
1020 FL=0
1030 IF (PEEK(REGDAT)AND 1)=0 THEN FL=1
1040 IF (PEEK(REGDAT)AND 1)<>1 THEN 1040
1050 RETURN
1999 :
2000 REM S/R SUMAR
2010 IF FL=1 THEN CC=CC+1
2020 RETURN
2999 :
3000 REM S/R VISUALIZACION
3010 POKERDD,255
3020 POKEREGDAT,CC
3030 RETURN
```

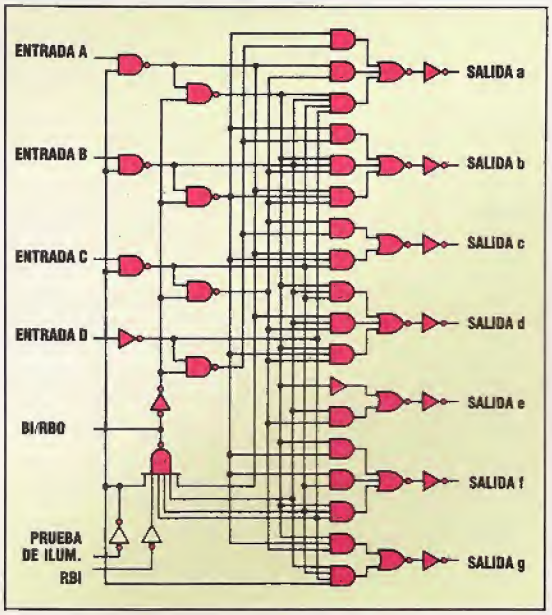
En cada ciclo del programa se utiliza un bucle FOR-...NEXT para repetir muchas veces la ejecución de la rutina en la cual se establecen todas las líneas en salida, para cada vez que se ejecuta la rutina donde se establece la línea 0 en entrada. En la versión para el Commodore 64, se ejecutan 20 rutinas de visualización por cada rutina de entrada. Esta proporción se incrementa a 40 en la versión para el BBC, debido a la mayor velocidad de ejecución del BBC Micro. Con estas proporciones, aún sigue detectándose un centelleo, pero si se incrementara esta proporción podría suceder que el tiempo de espera de una entrada se redujera tanto que las entradas llegaran a perderse.

Todo empaquetado
Dado que cada visualización digital necesita un código de entrada de cuatro bits, se pueden activar dos visualizaciones desde la puerta para el usuario. Éstas se colocan dentro de una unidad compatible con interfaces y dispositivos

Decimal	Binario				Salida	Visual.
	D3	D2	D1	D0	a b c d e f g	
0	0	0	0	0	0000001	0
1	0	0	0	1	1001111	1
2	0	0	1	0	0010010	2
3	0	0	1	1	0000110	3
4	0	1	0	0	1001100	4
5	0	1	0	1	0100100	5
6	0	1	1	0	1100000	6
7	0	1	1	1	0001111	7
8	1	0	0	0	0000000	8
9	1	0	0	1	0001100	9
10	1	0	1	0	1110010	c
11	1	0	1	1	1100110	ɔ
12	1	1	0	0	1011100	u
13	1	1	0	1	0110100	S
14	1	1	1	0	1110000	E
15	1	1	1	1	1111111	

Calculando
La entrada a la visualización digital desde la puerta para el usuario es un número binario de cuatro bits. Relacionado con cada uno de los números del 0000 al 1111 hay un número exclusivo de siete bits, señalando cada uno de ellos el estado de uno de los siete segmentos de la visualización. Según este código de visualización, un bit cero significa que se ha de encender el segmento correspondiente, y un uno que el segmento correspondiente debe permanecer apagado

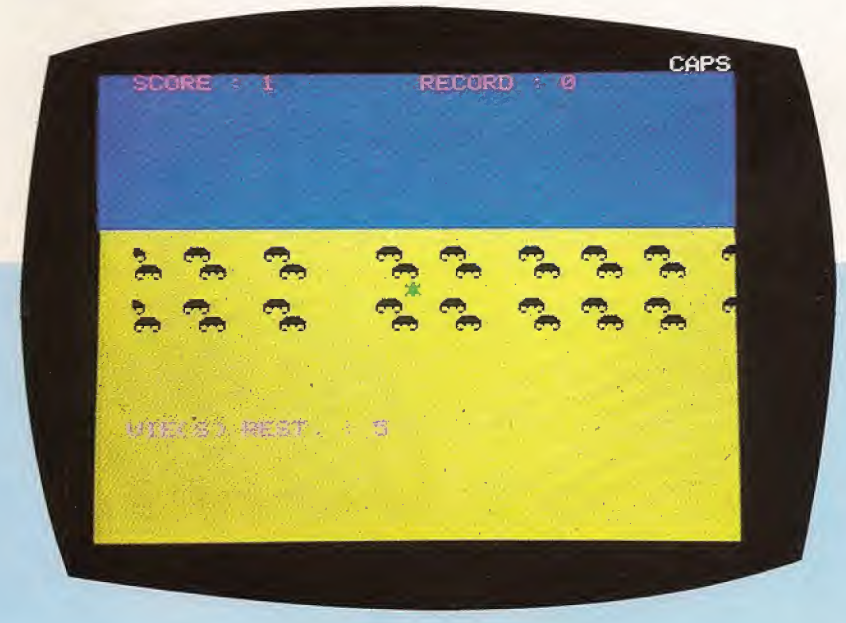




Decodificador/activador BCD 7447A a siete segmentos
El sistema de circuitos interno del chip muestra la sencillez esencial de su lógica: la entrada de cuatro bits se decodifica en salida para siete segmentos mediante puertas lógicas. La entrada de la prueba de iluminación enciende simultáneamente todos los segmentos para probar el chip

Cangrejos

Las leyes del mar revisadas y corregidas en función de la informática. He aquí un atractivo juego para el Oric Atmos



El usuario debe ayudar a una pobre tortuga a volver al mar, evitando a los voraces cangrejos que deambulan por la playa. Cada tortuga que alcance su propósito le proporciona un punto. Dispone de cinco itinerarios para intentar conseguir su puntuación máxima. Emplee las teclas W para avanzar y Z para retroceder.

```
10 REM **************
20 REM * CANGREJOS *
30 REM **************
35 POKE #26A,PEEK (#26A) OR 8
40 PRINT CHR$(17)
50 GOSUB 840
60 PLOT 2,20,"VIDA(S) REST. :"+STR$(NP)
90 A$=RIGHT$(A$,1)+LEFT$(A$,37)
100 B$=RIGHT$(B$,37)+LEFT$(B$,1)
120 PLOT 2,X1,A$
140 PLOT 2,X2,B$
160 PLOT 2,X3,A$
180 PLOT 2,X4,B$
190 D$=KEY$
200 PY=PY+(D$="W")-(D$="Z")
210 IF PY>16 THEN PY=16
220 IF PY=8 THEN 360
230 C=SCRN(PX,PY)
240 IF C<>32 AND C<>91 THEN 550
260 PLOT PX,YP,N$
280 PLOT PX,PY,P$
300 YP=PY
320 T=T+1
330 IF T>500 THEN 660
340 GOTO 60
360 PLOT PX,YP,N$
380 PLOT PX,PY,P$
420 GOSUB 1600
460 PY=16
470 YP=PY
480 S=S+1
500 PLOT 2,0,"PUNTOS :"+STR$(S)
520 PLOT 20,0,"RECORD :"+STR$(R)
530 GOSUB 1110
540 GOTO 60
550 NP=NP-1
570 PLOT PX,YP,N$
580 PLOT PX,PY,"*"
600 GOSUB 1500
610 IF NP=0 THEN 660
620 PY=16
630 YP=PY
640 GOSUB 1110
650 GOTO 60
660 CLS
665 INK 0
670 IF S>R THEN R=S
680 IF T<500 THEN 720
690 PLOT 8,6,"**TIEMPO TRANSCURRIDO**"
720 PLOT 13,10,"PUNTOS:"+STR$(S)
740 PLOT 12,14,"RECORD :"+STR$(R)
760 PLOT 11,20,"OTRA ?"
770 REPEAT
780 D$=KEY$
790 UNTIL D$=""
795 REPEAT
800 D$=KEY$
805 UNTIL D$<>""
810 IF D$ "N" THEN S0
815 CLS
820 PAPER 7
825 INK 0
830 PRINT CHR$(17)
835 END
840 CLS
850 PAPER 3
860 INK 2
870 RESTORE
880 FOR I=0 TO 23
890 READ A
900 POKE 46808+I,A
910 NEXT I
920 P$=CHR$(91)
930 N$=CHR$(32)
940 A$=""
950 B$=""
960 S=0
970 NP=5
980 PX=19
990 PY=16
1000 YP=PY
1010 X1=10
1020 X2=11
1030 X3=13
1040 X4=14
1050 T=0
1060 FOR I=1 TO 38
1070 READ A
1080 A$=A$+CHR$(A)
1090 NEXT I
1100 B$=A$
1110 X=INT(RND(1)*36)+1
1120 A$=RIGHT$(A$,X)+LEFT$(A$,38-X)
1140 PLOT 1,20,CHR$(5)
1150 PLOT 1,X1,CHR$(0)
1160 PLOT 1,X2,CHR$(0)
1170 PLOT 1,X3,CHR$(0)
1180 PLOT 1,X4,CHR$(0)
1190 PLOT 0,0,CHR$(18)
1200 PLOT 1,0,CHR$(5)
1210 FOR X=0 TO X1-2
1220 PLOT 0,X,CHR$(20)
1230 NEXT X
1240 RETURN
1500 ZAP
1510 RETURN
1520 WAIT 300
1530 RETURN
1600 PING
1610 FOR PY=PY TO 0 STEP-1
1620 PLOT PX,YP,N$
1630 PLOT PX,PY,P$
1635 WAIT 30
1640 YP=PY
1650 NEXT PY
1660 PLOT PX,YP,N$
1670 WAIT 100
1680 RETURN
2000 DATA 12,45,63,30,30,63,45,0
2010 DATA 7,15,31,31,18,16,12,0
2020 DATA 56,60,62,62,18,2,12,0
2030 DATA 32,92,93,32,32,92,93,32,32,32,
     92,93,32,32,32,32,92,93,32
2040 DATA 32,92,93,32,32,32,92,93,32,32,
     92,93,32,32,92,93,32,32,32
```

De compras

Establecemos una comparación entre algunos de los ordenadores más populares, destacando sus cualidades y sus puntos débiles

Ian McKinnell

A la hora de adquirir una nueva máquina es importante considerar cuáles son exactamente las necesidades del futuro usuario: ¿desea un ordenador que se pueda ampliar mediante la adición de periféricos, memoria extra, etc., o puede permitirse tratarlo como un producto desechable, para vendérselo a otra persona cuando surja algo mejor?

La mayoría de los nuevos modelos ofrecen más facilidades que sus rivales más antiguos, incluyendo memorias más grandes, mejores versiones de BASIC, gráficos en resolución más alta y software incorporado. Pero las máquinas más antiguas, en especial aquellas de las cuales se han vendido grandes cantidades, poseen una ventaja fundamental: disponibilidad de software. Muchos compradores de ordenadores más nuevos habrán de esperar durante meses antes de disponer de una gran gama de software; y, en algunos casos, éste no aparecerá nunca. En este sentido, el Oric Atmos es un buen ejemplo. Esta versión mejorada del Oric-1 ha estado a la venta durante meses, pero los escritores de software se han mostrado reacios a producir material para el mismo. Como resultado de esto, las cifras de ventas de esta máquina han descendido drásticamente.

Los tres micros que están mejor servidos por las casas de software son el Sinclair Spectrum, el Commodore 64 y el BBC Micro. El Spectrum, en particular, es un clásico ejemplo de la forma en que la escritura de software creativo puede superar las limitaciones intrínsecas de una máquina: algunos de los programas para este micro se pueden comparar muy favorablemente con aquellos producidos para máquinas considerablemente más sofisticadas. Sin embargo, es poco probable que alguno de estos tres ordenadores se vendiera bien si se los lanzara al mercado de hoy en día: el Spectrum tiene un teclado sumamente pobre, el BASIC del Commodore 64 carece de las instrucciones que permitirían aprovechar al máximo el potencial de la máquina, y el BBC Micro tiene una memoria pequeña y su precio es excesivo para los estándares actuales.

La mayor parte de los micros más recientes poseen especificaciones más atractivas, pero carecen de la profundidad y amplitud de software. Cualquiera que adquiera una de estas máquinas está apostando a favor de que obtenga popularidad y, que, por tanto, convenza a quienes desarrollan software para que creen programas para la misma.

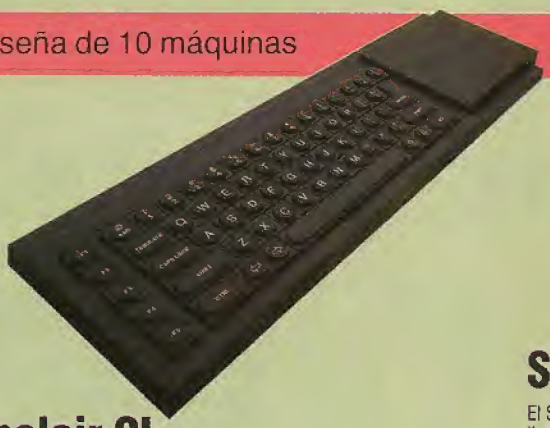
La tendencia principal en el caso de los ordenadores personales nuevos es ofrecer más por el mismo dinero. Los teclados de gran calidad, las memorias más grandes para el usuario (64 Kbytes o más) y los buenos gráficos en la actualidad ya son algo estándar. La calidad del intérprete de BASIC se ha mejorado considerablemente en máquinas como el Commodore Plus/4, Commodore 16, Sinclair QL y los micros MSX. El Amstrad incluye en su precio hasta una pantalla monocromática o en color.

Otra interesante tendencia actual es la inclusión de software "empaquetado" o gratuito. El Sinclair QL se suministra con cuatro programas de esta clase: un paquete de tratamiento de textos, hoja electrónica, base de datos y gráficos de gestión. El Commodore Plus/4 proporciona una gama similar, si bien los programas son menos sofisticados y, en realidad, requieren una unidad de disco para poder utilizarlos. Otros micros se concentran en los juegos. Con el Commodore 16 se suministran cuatro juegos, y hasta Sinclair ha empezado a proporcionar un paquete de seis juegos con su ya un tanto anticuado Spectrum.

Quien adquiera un micro nuevo deberá considerar también otros puntos. Algunas máquinas son más ampliables que otras, permitiendo la utilización de unidades de disco, impresoras, modems y otros periféricos. Algunos ordenadores aceptarán accesorios estándares, mientras que otros exigirán periféricos de su "propia marca", lo que limita las opciones del usuario. Un buen manual es esencial: algunas máquinas se entregan con manuales deficientes, que en lugar de aclarar las cosas lo que hacen es confundir. El comprador en cierne también deberá considerar el tipo de software disponible para cada máquina; por ejemplo, el BBC Micro posee una elevada proporción de software educativo, mientras que el Spectrum es una opción mejor si de juegos se trata.

Para una decisión acertada

Comprar un ordenador debería ser tan fácil como adquirir un traje o un vestido. Sin embargo, el cúmulo de información técnica y la amplia gama de opciones que se le presentan al usuario convierten el acto de decidirse en una auténtica tómbola. Lo primordial antes de visitar la tienda de ordenadores es comparar ecuánime y desapasionadamente lo que usted en realidad necesita y la capacidad y características de las diferentes máquinas. Trate de decidir con antelación qué es lo que va a comprar, y deje que la "atracción" de la máquina escogida sea el último —nunca el primero— factor decisorio



Amstrad CPC 464

Amstrad es una empresa que posee una rica experiencia en el mercado de alta fidelidad, y ésta se refleja en el diseño compacto de su primer micro personal (véase p. 909). El hecho de que la máquina se suministre con una pantalla y con una grabadora de cassette incorporadas la hará muy atractiva para el usuario novel, y el Amstrad configura interfaces tanto para palanca de mando como Centronics. Existe, asimismo, una unidad de disco que se vende junto con el lenguaje LOGO y el sistema operativo CP/M. El Amstrad es una máquina buena y completa, con una resolución para gráficos máxima de 640 por 200 pixels en dos colores, con una "paleta" de 27 tonalidades y facilidades para sonido estéreo

Sinclair QL

El QL se vende con 128 K de memoria, microdrives gemelos incorporados y cuatro programas de gestión empaquetados (véase p. 981). En vista de lo cual, parece ofrecer unas prestaciones notables; pero existen limitaciones. El teclado es decepcionante: es sólo una versión más sofisticada del teclado estilo membrana utilizado en el Spectrum; el BASIC es estructurado, pero contiene algunos errores y es sorprendentemente lento; la fiabilidad a largo plazo del microdrive es, asimismo, dudosa. Las facilidades de edición también son decepcionantes.

El QL soporta una resolución para gráficos de 512 por 256 pixels en cuatro colores, o de 256 por 256 en ocho. Cada pixel se puede colorear de forma individual, de modo que se utilizan 32 K de memoria sólo para manipular la pantalla. No se suministra interface para disco ni para cassette, pero el QL posee una interface para palanca de mando, otra para pantalla, una tercera para conexión en red y dos RS232. Es probable que uno de los principales reclamos de cara a las ventas sea el software empaquetado, escrito por Psion. Estos cuatro programas (tratamiento de textos, base de datos, hoja electrónica y programas para gráficos de gestión) son muy sofisticados en comparación con el software de otras máquinas personales. No obstante, se ven deslucidos por las limitaciones del hardware

Sinclair Spectrum

El Spectrum se ha convertido en un éxito a pesar de sus limitaciones (véase p. 530). El teclado es muy pobre y el ordenador utiliza un sistema de "palabra-tecla" para la entrada de programas, lo que hace que la programación le resulte más sencilla al principiante pero que le plantee problemas al usuario más experimentado. La resolución de la pantalla es de 256 por 176 pixels con ocho colores, dos de los cuales se pueden utilizar en cualquier posición de carácter. La facilidad de sonido es prácticamente inexistente, con una única "voz" y un nivel de volumen virtualmente inaudible. El BASIC es aceptable, aunque algo lento, pero el manual cumple bien su función de enseñar el BASIC. El Spectrum carece por completo de interfaces estándares, si bien muchas firmas independientes han producido equipos de periféricos que simplemente se "enganchan" en la puerta para el usuario de la máquina. Recientemente Sinclair ha producido su propia Interface 1, que soporta microdrives, conexión en red y un enlace RS232. Ésta fue rápidamente seguida por la Interface 2, que le confirió a la máquina capacidad para utilizar software en cartucho. Aunque algo anticuada, la base de software del Spectrum lo convierte en una propuesta atractiva, en especial porque con la versión de 48 K se suministran de forma gratuita seis programas

BBC Modelo B

Esta máquina se utiliza bastante en los centros docentes; por este motivo puede resultarles familiar a los usuarios más jóvenes; existe una buena gama de software educativo (véase p. 929). Fabricado por Acorn, las especificaciones del BBC Micro son excelentes, con un BASIC "estructurado" muy veloz, excelente resolución para gráficos, buen sonido y una notable gama de interfaces, incluyendo Centronics, RS423, RGB, video compuesto, cuatro canales de A/D, una puerta para el usuario, bus de 1 MHz para accesorios y el "tubo", que permite conectar a la máquina un segundo procesador. También soporta unidades de disco y conexión en red. No obstante, la relativamente pequeña RAM para el usuario es devorada rápidamente por los gráficos, dejando entre 9 y 28 K para el usuario, dependiendo de la modalidad seleccionada. La utilización de un segundo procesador atenúa este problema, y la selección de la opción Z80 para segundo procesador permite la ejecución de software CP/M, haciendo al BBC apto para su empleo como máquina de gestión

Acorn Electron

El Electron es una versión a escala reducida del BBC Micro; posee el mismo y excelente BASIC estructurado, pero carece de la amplia gama de interfaces de que dispone el BBC (véase p. 929). El BASIC Electron trabaja a una velocidad inferior a la de la versión BBC, y el Electron no posee la modalidad 7 del BBC para gráficos de teletexto. Parte del software para el BBC es compatible con el Electron, mientras que existen otros programas que se han escrito especialmente para él. El Electron puede producir visualizaciones notables, con una resolución máxima para gráficos de 640 por 256 pixels, y una resolución máxima para textos de 32 líneas por 80 caracteres. Lamentablemente, esto reduce la cantidad de memoria disponible para el programador: del máximo de 32 K, al usuario apenas le quedan 9 K si selecciona la resolución más alta. El periférico Plus 1 proporciona las interfaces para impresora, palanca de mando y cartucho que no vienen con la máquina básica, pero hasta ahora no hay ninguna unidad de disco disponible

Para comprar un micro

La compra de un nuevo micro puede ser una fuente de confusión. Nuestro gráfico especifica las características más importantes de un micro y da perfiles comparativos para las máquinas incluidas en este capítulo. Todo lo que debe hacer es comparar sus necesidades con estos perfiles

Amstrad CPC 464

Pobre			Bueno
			Precio
			Memoria
			Almac. de apoyo
			Calidad teclado
			Gráficos BASIC
			Sonido BASIC
			Editor BASIC
			Facilidades BASIC
			Calidad software
			Cantidad software
			Interfaces
			Salida a pantalla

Sinclair QL

Pobre			Bueno
			Precio
			Memoria
			Almac. de apoyo
			Calidad teclado
			Gráficos BASIC
			Sonido BASIC
			Editor BASIC
			Facilidades BASIC
			Calidad software
			Cantidad software
			Interfaces
			Salida a pantalla

Sinclair Spectrum

Pobre			Bueno
			Precio
			Memoria
			Almac. de apoyo
			Calidad teclado
			Gráficos BASIC
			Sonido BASIC
			Editor BASIC
			Facilidades BASIC
			Calidad software
			Cantidad software
			Interfaces
			Salida a pantalla

BBC Modelo B

Pobre			Bueno
			Precio
			Memoria
			Almac. de apoyo
			Calidad teclado
			Gráficos BASIC
			Sonido BASIC
			Editor BASIC
			Facilidades BASIC
			Calidad software
			Cantidad software
			Interfaces
			Salida a pantalla

Acorn Electron

Pobre			Bueno
			Precio
			Memoria
			Almac. de apoyo
			Calidad teclado
			Gráficos BASIC
			Sonido BASIC
			Editor BASIC
			Facilidades BASIC
			Calidad software
			Cantidad software
			Interfaces
			Salida a pantalla

Commodore Plus/4

Ésta es la máquina que a la larga podría sustituir al Commodore 64. Ambos ordenadores poseen una resolución para gráficos similar de 320 por 200 pixels y 64 K de RAM, pero el Plus/4 puede visualizar 121 colores y tiene un BASIC muy mejorado, lo que le proporciona al usuario mayor control sobre la visualización en pantalla. En la modalidad de resolución más alta, sólo se pueden visualizar dos colores en cada posición de carácter, pero la selección de la modalidad de 160 por 200 pixels permite que sean cuatro por posición. El sonido no está del todo a la altura de los elevados estándares del C64, con un máximo de dos "voces"; sin embargo, su superior BASIC hace que la manipulación de sonido resulte mucho más fácil. En el nuevo modelo se ha omitido la facilidad para gráficos sprite del C64. El micro lleva incorporado un monitor de código máquina, así como cuatro pequeños programas "serios": tratamiento de textos, hoja electrónica, base de datos y programa para gráficos. Sin embargo, para aprovechar al máximo los mismos es necesaria una unidad de disco. El Plus/4, de acuerdo a la política de Commodore, exige su propia grabadora de cassette, pero éste no es del mismo modelo que utilizan el Vic y el Commodore 64. También se requieren palancas de mando especiales. El Plus/4 emplea las mismas impresoras y la misma lenta unidad de disco que el C64, si bien se está desarrollando una unidad más rápida

Commodore 64

Micro bien establecido en el mercado, con gran riqueza de software disponible, el 64 adolece de un BASIC pobre, que carece de instrucciones incorporadas para sacar partido de los excelentes gráficos y sonido (véase p. 490). La resolución máxima es de 320 por 200 pixels con 16 colores en pantalla, si bien en cada posición de carácter sólo se pueden visualizar dos colores. También posee sprites. A pesar de los 64 K de RAM, no hay más de 39 K disponibles para el usuario. El C64 exige una grabadora de cassette especial y, si se desea una ampliación, son necesarias impresoras y unidades de disco de la misma marca. La unidad de disco es bastante lenta en operación y muy sujeta a errores

Commodore 16

Diseñado para reemplazar al ya superado Vic-20, el Commodore 16 se suministra en un "paquete de iniciación" que contiene grabadora de cassette, cinta y libro para aprendizaje de BASIC y cuatro programas "recreativos". Si bien la carcasa le proporciona un aspecto similar al Commodore 64 y al Vic-20, por dentro la máquina se aproxima mucho más al Plus/4, ya que utiliza el mismo BASIC y el monitor de código máquina. En uso normal, quedan 12 K para empleo en BASIC, si bien los gráficos en alta resolución reducirán esta cantidad a unos insuficientes 2 K. Esta escasa asignación de memoria significa que la mayor parte del software para el 16 se encuentra en cartucho. De momento es muy poco el software que ha aparecido, pero los programas que se proporcionan con ella incluyen dos juegos recreativos, un programa de ajedrez que a la vez lo enseña, y un programa para diseñar gráficos. El C16 ofrece una buena relación calidad-precio, en especial para el novato, pero la pequeña cantidad de memoria libre podría plantearles problemas a los escritores de software

Atari 600/800 XL

Estas máquinas son versiones mejoradas de la antigua serie Atari 400/800 (véase p. 669). Esto significa que hay una amplia gama de software disponible, si bien los ordenadores ahora parecen un poco anticuados. El Atari 600XL gradualmente está dejando de producirse, pero continúa siendo una buena adquisición mientras existan stocks, dado que sus 16 K de RAM se pueden ampliar a 64, convirtiéndolo de hecho en un 800 XL. La resolución máxima para gráficos es de 320 por 192 pixels, si bien en esta modalidad sólo se pueden visualizar dos colores. La selección de una resolución más baja otorga 16 colores en 16 tonalidades diferentes. Otras características notables son un sonido y unos gráficos sprite sobresalientes, aunque el BASIC que utiliza Atari en la actualidad está algo desfasado. Las máquinas Atari exigen una reproductora de cassette exclusiva, que implica un gasto adicional. Es de lamentar que, junto con los micros, no se suministre un manual completo; éste se debe adquirir por separado. Los periféricos Atari no son estándares, pero suele ser muy fácil conseguirlos y su precio es razonable

MSX Estándar

El MSX es un "estándar mínimo" y muchos fabricantes ofrecerán más que las especificaciones básicas, si bien ninguna de las mejoras afectará a la compatibilidad. La máquina de nuestra ilustración es el Toshiba HX-10 (véase p. 1149). El estándar MSX especifica una versión particularmente buena de BASIC, que incluye instrucciones para gráficos, sonido y tratamiento de eventos muy fáciles de utilizar, así como un buen editor. Las máquinas MSX poseen teclas de función, que se pueden programar para 10 funciones o instrucciones diferentes. La visualización da una resolución para gráficos de 256 por 192 pixels en 16 colores; también hay 32 sprites disponibles. Para hacer frente a la visualización, de los 80 K de RAM hay 16 reservados para la manipulación de pantalla. De los 64 restantes, el programador puede emplear 28 K; para acceder a los demás es necesario utilizar código máquina o bien una unidad de disco

Descenso de los precios
Hemos preferido no incluir los precios de los ordenadores, dado que están variando de manera constante. Se advierte una significativa tendencia al descenso en los últimos meses, siendo éste, a veces, drástico

Commodore Plus/4

Pobre			Bueno
			Precio
			Memoria
			Almac. de apoyo
			Calidad teclado
			Gráficos BASIC
			Sonido BASIC
			Editor BASIC
			Facilidades BASIC
			Calidad software
			Cantidad software
			Interfaces
			Salida a pantalla

Commodore 64

Pobre			Bueno
			Precio
			Memoria
			Almac. de apoyo
			Calidad teclado
			Gráficos BASIC
			Sonido BASIC
			Editor BASIC
			Facilidades BASIC
			Calidad software
			Cantidad software
			Interfaces
			Salida a pantalla

Commodore 16

Pobre			Bueno
			Precio
			Memoria
			Almac. de apoyo
			Calidad teclado
			Gráficos BASIC
			Sonido BASIC
			Editor BASIC
			Facilidades BASIC
			Calidad software
			Cantidad software
			Interfaces
			Salida a pantalla

Atari 600/800 XL

Pobre			Bueno
			Precio
			Memoria
			Almac. de apoyo
			Calidad teclado
			Gráficos BASIC
			Sonido BASIC
			Editor BASIC
			Facilidades BASIC
			Calidad software
			Cantidad software
			Interfaces
			Salida a pantalla

MSX Estándar

Pobre			Bueno
			Precio
			Memoria
			Almac. de apoyo
			Calidad teclado
			Gráficos BASIC
			Sonido BASIC
			Editor BASIC
			Facilidades BASIC
			Calidad software
			Cantidad software
			Interfaces
			Salida a pantalla

Máquina de calcular

He aquí una serie cuya finalidad es ser una guía práctica en el uso de hojas electrónicas basadas en cassette

Existen en el mercado varios paquetes de modelos financieros, basados en cassette, para ordenadores personales populares como el Sinclair Spectrum, Commodore 64 y BBC Modelo B. Esta serie de capítulos pretende proporcionar una guía práctica, paso a paso, del empleo de tales paquetes de hoja electrónica para numerosas aplicaciones cotidianas, entre las que se incluyen presupuesto doméstico, cálculo de la incidencia que podría tener cualquier aumento de los tipos de interés en los pagos de hipotecas, y comparación de los valores relativos de alquilar o comprar artículos para el hogar.

El corazón de todos los paquetes de hoja electrónica es una "hoja de trabajo" electrónica que se divide en filas y columnas (parecida a una gran hoja pautada de papel cuadriculado para gráficas y diseño). La pantalla de televisión (o de ordenador) actúa como una ventana móvil que puede visualizar cualquier porción de esta hoja (que es mayor que las cuatro o cinco columnas y las 10 o 15 filas que aparecen en pantalla en un momento dado).

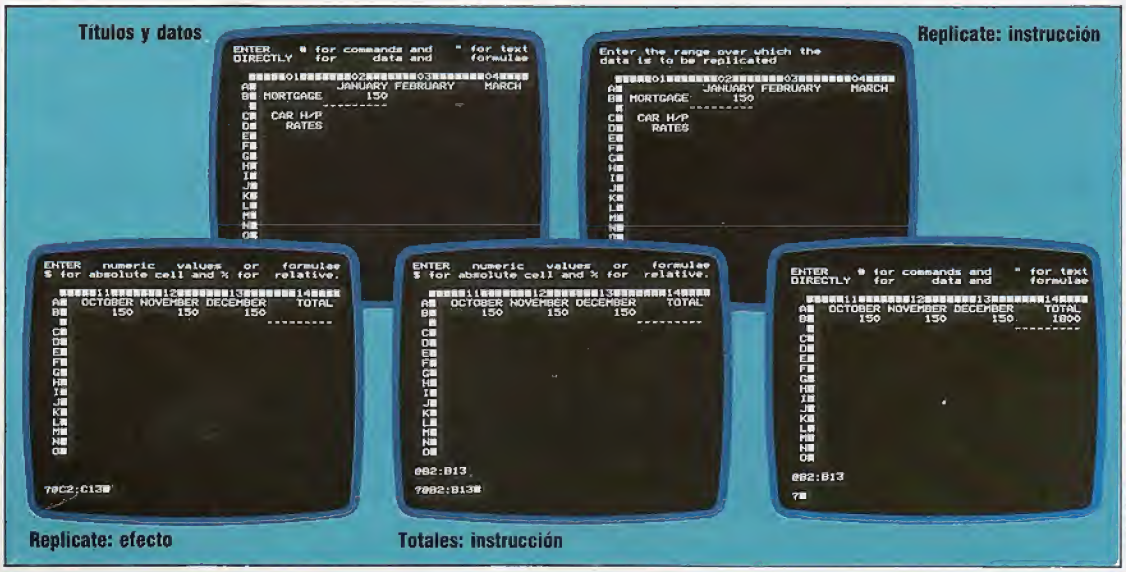
La intersección de una columna y una fila se denomina *celda*: cada celda puede contener números, texto o fórmulas. La hoja electrónica utiliza un cursor (normalmente un bloque realzado) que se puede desplazar a través de ella mediante las teclas para control del cursor. El programa supone que toda entrada de datos desde el teclado está destinada a la celda que en ese momento ocupa el cursor.

Éste es un esquema muy general del diseño básico de una hoja electrónica. En este primer capítulo nos concentraremos en un paquete llamado *Vu-Calc*; lo comercializa Psion y se vende tanto para el Spectrum como para el BBC Modelo B. Aquí vamos a considerar la versión para el BBC, que es casi idéntica (a excepción de ciertas diferencias, irrelevantes pero molestas, en los nombres de algunas instrucciones) a la versión para el Spectrum.

El *Vu-Calc* demuestra lo flexible y lo útil que puede ser una hoja electrónica sencilla. No posee ninguna de las características más sofisticadas de paquetes de modelización como el *Lotus 1-2-3* (véase p. 1124). Uno no puede dividir la pantalla ni horizontal ni verticalmente para mostrar distintas porciones de la hoja de trabajo al mismo tiempo (una facilidad de que disponen los paquetes más "serios"), y tampoco se dispone de la cantidad de facilidades que ofrecen las hojas electrónicas diseñadas con fines de gestión. Pero se puede utilizar el *Vu-Calc* para construir algunos modelos muy útiles, que se pueden guardar en cassette, junto con sus datos, para referencia posterior.

La versión del *Vuc-Calc* para el BBC Micro posee un máximo de 28 columnas (numeradas del 1 en adelante) y 52 filas (etiquetadas alfabéticamente, con las filas después de la "Z" etiquetadas con letras dobles: "AA", "BB", etc.). Este tamaño no es muy grande de acuerdo a los estándares de hojas electrónicas, pero el límite obedece a la más bien escasa memoria de 32 K del BBC Micro.

Una de las aplicaciones más útiles para una hoja electrónica de ordenador personal es el presupuesto doméstico anual, que posee asimismo el mérito de ser un modelo de construcción relativamente simple. Una vez construido, un modelo de este tipo permite ver a simple vista qué impacto tendrá el aumento de cualquier gasto (p. ej., una cuenta del teléfono excesivamente elevada o un viaje al extranjero que no estaba previsto) sobre el superávit que se esperaba (en el supuesto, claro, de que esperara alguno). En otras palabras, una vez que el modelo está construido, se puede jugar con los datos y



Ian McKinnell

ver cómo se reflejará el efecto de una alteración a través de la hoja electrónica como un todo.

Construyendo un modelo

El primer paso para construir tal modelo consiste en escribir en un trozo de papel una lista de todos los gastos domésticos que se le ocurran. La siguiente tarea es anotar debajo de cada título la cifra mensual esperada. Es aquí, en el punto de calcular o estimar la cantidad mensual y entrarla en el modelo, donde se hacen evidentes los puntos fuertes del *Vu-Calc*. Las categorías de gastos se digitan una tras otra a lo largo de la primera columna, y cada columna subsiguiente se etiqueta "Ene", "Feb", "Mar", etc. El *Vu-Calc* exige que todas las entradas de texto vayan precedidas por comillas, pero en la hoja electrónica visualizada este signo de puntuación desaparece. Las primeras filas y columnas de nuestro modelo, por consiguiente, podrían ser así:

	1	2	3	4
A		ENE	FEB	MAR
B	Coche			
C	Hipoteca	25 000		
D	Impuestos			

Esto nos proporciona la "forma" básica del modelo. Ahora necesitamos entrar los valores correspondientes, y para hacer esto utilizamos la instrucción REPLICATE del *Vu-Calc*.

El *Vu-Calc* posee una cantidad limitada de instrucciones, todas las cuales llevan el prefijo de un signo #. (Cuando se entra # en una celda vacía, el programa pasa a la "modalidad instrucción" y espera a que se le diga cuál es la instrucción que se desea emplear.) La más útil de estas instrucciones es REPLICATE, porque la parte más aburrida de la construcción de un modelo es la necesidad de digitar todos los valores que utilizará el mismo. REPLICATE es básicamente un dispositivo ahorrador de trabajo que permite entrar el mismo dato en muchas celdas diferentes de forma simultánea.

Una parte común del modelo de presupuesto/gastos domésticos son los gastos mensuales fijos, como los impuestos, la hipoteca o el alquiler. Si la hipoteca es, por ejemplo, 25 000 ptas por mes, se utilizará la instrucción REPLICATE para insertar esta cantidad en las 12 celdas adecuadas.

Cuando se invoca la instrucción REPLICATE, digitando #R, en la parte superior de la pantalla, por encima del modelo propiamente dicho, aparece la siguiente línea de indicación:

> Replicate - Enter the cell to replicate, RETURN for the current cell. (Replicate - Entre la celda a reproducir, RETURN para la celda en curso.)

Ésta es una indicación para que se especifiquen las celdas que se han de copiar (observe que puede copiar una sola celda, no un bloque entero de celdas; ésta es una de las limitaciones más evidentes del *Vu-Calc*). La celda se especifica mediante sus coordenadas, con la letra de la fila en primer lugar, seguida del número de columna; por ejemplo: C2. Después de entrada la celda, la línea indicadora le pide que:

> Enter the range over which the data is to be replicated. (Entre la serie en la cual se ha de reproducir el dato.)

Una serie de celdas se indica en *Vu-Calc* especificando la primera celda (o la situada más a la izquierda) de la serie y la celda situada más abajo y más a la derecha de la serie. (Imagine que el bloque de celdas es como un recuadro: hay que decirle al programa las coordenadas de las esquinas superior izquierda e inferior derecha del recuadro.)

En nuestro ejemplo, deseamos decirle al programa que coloque 25 000 en la serie de celdas de la Fila C, desde C3 hasta C13 (de la columna etiquetada "Feb" a la etiquetada "Dic"). Para esto el formato es #R,C2,C3:C13. Con esto se rellena automáticamente cada celda, casi al instante, con el valor 25 000. (El verdadero poder de la instrucción REPLICATE es copiar fórmulas de una celda a otra, pero ésta es un área bastante especializada, puesto que tales fórmulas pueden ser "relativas" respecto a una celda determinada o bien "absolutas", distinción y tema estos que abordaremos pronto.)

Ahora trabajamos con todos nuestros títulos de gastos de la misma forma. Observe que si decide que en según qué meses la cantidad para un gasto determinado debe ser mayor o menor que el valor estándar, puede simplemente mover el cursor hasta esa celda y entrar un nuevo valor. Éste se escribirá de inmediato sobre la cifra anterior.

En nuestro modelo, la columna 14 será la columna de totales anuales. Tendría poco sentido utilizar una hoja electrónica si se tuviera que usar una calculadora para ir sumando los gastos de cada mes, de modo que el *Vu-Calc* se puede utilizar para sumar todos los valores de cualquier fila o columna. La instrucción @ indica que se desea sumar los valores que contiene una serie de celdas; esta serie se especifica de la misma forma que antes. Por lo tanto, para totalizar los pagos de hipoteca de todo el año y colocar el resultado en la celda C14, sencillamente se pone el cursor en C14 y se digita @C2:C13. El resultado (300 000 en este caso) se visualiza inmediatamente.

En la celda C14 podríamos igualmente haber colocado una fórmula; en vez de sumar todos los valores podríamos haber entrado C2*12. *Vu-Calc* habría tomado esto como una fórmula, dado que la C no iba precedida por comillas, y la hubiera ejecutado de inmediato, dando el resultado 300 000. Esto sirve para ilustrar que a menudo hay más de una manera de conseguir un resultado determinado.

En el próximo capítulo del curso analizaremos cómo los gastos pueden "crecer" en función de porcentajes fijos, y cómo se pueden reproducir fórmulas de referencia de celdas relativas y absolutas.

Desplegando la hoja

Commodore 64

Busicalc: cassette/disco por Supersoft, Canning Road, Harrow HA3 7SJ, Gran Bretaña.

Insta-Calc Graphic: cartucho/disco por Dataview Wordcraft Ltd., Radix House, East Street, Colchester CO1 2XB, Gran Bretaña.

BBC Micro

Vu-Calc: cassette por Psion Ltd., 2 Hunstworth Mews, Gloucester Place, London NW1 6DD, G.B.

Spectrum

Vu-Calc: cassette por Psion Ltd., Gran Bretaña.

Vencedor del dragón

Continuando con el estudio de los sprites en LOGO, desarrollaremos el algoritmo de la "curva de persecución"

Aquí ofrecemos los procedimientos para un juego que utiliza sprites de LOGO. El jugador controla a un dragón que intenta llegar hasta una ciudad y destruirla. La defensa de la ciudad está en manos de un caballero volador (controlado por el ordenador), quien intentará matar al dragón. El jugador controla la dirección de movimiento del animal mediante la palanca de mando. Si se burla al caballero y logra acercarse a la ciudad, ésta se verá envuelta en llamas debido al fuego exhalado por el dragón.

Para ejecutar el juego habrá de leerse el archivo SPRITES, definir sus formas, entrar los procedimientos y luego digitar JUEGO. Después de llevar a cabo diversas tareas de preparación, el procedimiento JUEGO llama a JUGAR, que es el procedimiento central. JUGAR hace mover al dragón y al caballero y verifica si el monstruo ha llegado a la ciudad o si el caballero le ha acertado. Los restantes procedimientos realizan otras partes de JUGAR.

Las instrucciones de color disponibles son muy directas. Para establecer el color de fondo utilice BACKGROUND seguido de un número de color, y para establecer el color de un sprite (y el color de la línea que trazará cuando tenga bajado su lápiz) utilice PENCOLOR. A los números de color se les asignan nombres en INIC.VARIABLES, de modo que podamos especificar colores mediante nombres, utilizando instrucciones tales como PENCOLOR :ROJO.

En el procedimiento JUGAR, la línea:

```
SI ACERTADO? THEN DRAGON.DESTRUIDO
```

se utiliza para comprobar si el caballero ha hecho blanco en el animal. El procedimiento ACERTADO? ilustra la forma en que podemos escribir en LOGO nuestras instrucciones de decisión. Devuelve un valor "TRUE (verdadero) o "FALSE (falso), y éste se emplea como una entrada para las sentencias IF. El resultado "TRUE haría entonces que se llevara a cabo la acción condicionada a la pregunta.

ACERTADO? utiliza un procedimiento del archivo

Caballeros y dragones

El juego en acción

La ciudad en llamas

El dragón vencido

Ian McKinnell

Caballero contra dragón

```
TO JUEGO
  INIC. VARIABLES
  PREPARAR.PANTALLA
  JUGAR
END

TO INIC.VARIABLES
  MAKE"LLAMA1
  MAKE"LLAMA1 2
  MAKE"DRAGON 3
  MAKE"CABALLERO 4
  MAKE"CIUDAD5
  MAKE"CIUDAD1 6
  MAKE"ROJO 2
  MAKE"NEGRO 0
  MAKE"AZUL 6
  MAKE"NARANJA 8
  MAKE"AMARILLO 7
END

TO PREPARAR.PANTALLA
  DRAW
  FULLSCREEN
  BACKGROUND :AZUL
  TELL 0
  PU
  HT
  TELL :LLAMA1
  HT
  HT
  TELL :LLAMA
  HT
  POSITION :DRAGON 100 100 :ROJO
  BIGX BIGY
  POSITION :CIUDAD (-52)(-80) :NEGRO
  BIGX BIGY
  POSITION :CIUDAD1 (-100)(-80) :NEGRO
  BIGX BIGY
  TELL 4
  PU
  POSITION :CABALLERO (100)(-100)
    :AMARILLO
  SMALLX SMALLY
END

TO JUGAR
  MOVIMIENTO.DRAGON
  IF DISTANCIA :DRAGON :CIUDAD<50 THEN
  CIUDAD. DESTRUIDA STOP
  IF DISTANCIA :DRAGON :CIUDAD1<50 THEN
  CIUDAD.DESTRUIDA STOP
  MOVIMIENTO.CABALLERO
  IF ACERTADO? THEN DRAGON.DESTRUIDO
  STOP
  JUGAR
END

TO MOVIMIENTO.DRAGON
  TELL :DRAGON
  MOVPALANCA JOYSTICK 1
  FD 10
END

TO MOVPALANCA :DIR
  IF :DIR<0 STOP
  SETH :DIR*45
END

TO DISTANCIA :A :B
  TELL :A
  MAKE"X1 XCOR
  MAKE"Y1 YCOR
```

SPRITES, TS?, que devuelve "TRUE si hay un sprite tocando al sprite en curso. ACERTADO? pone al dragón como sprite en curso y después pregunta si hay algo tocándolo.

La instrucción JOYSTICK toma 0 o 1 como entrada (que corresponden a las puertas 1 y 2). La salida es −1 si la palanca de mando está en el centro, 0 si está arriba, 1 si está en 45°, 2 si está en 90°, y así sucesivamente hasta 7. Aquí simplemente establecemos el encabezamiento del dragón en 45° multiplicado por el número de salida.

Mediante la utilización de sprites es fácil conseguir explosiones y efectos similares. Encima del objeto a destruir aparece de modo intermitente una forma que representa la explosión. Al sprite LLAMA le damos un número bajo para que tenga alta prioridad y aparezca encima de los otros sprites.

El ordenador controla al caballero, pero emplea una estrategia defensiva muy simple: el caballero se encabeza directamente hacia el dragón. Tal como está el juego, el monstruo puede escabullirse por el lado del caballero y causar estragos.

¿Cómo podemos mejorar la estrategia defensiva del caballero? Una forma sencilla consiste en aumentar su velocidad; con sólo incrementarla de 10 a 11 al dragón le es mucho más difícil poder escabullirse. (¡Salir de la pantalla es hacer trampa!). Una estrategia acertada sería cortarle el paso al dragón orientando su encabezamiento hacia la línea entre éste y la ciudad y quedándose allí.

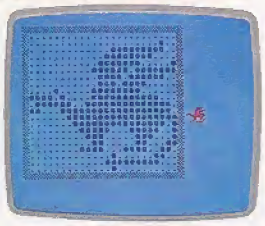
Con tres insectos

Los insectos parten de las esquinas de un triángulo:

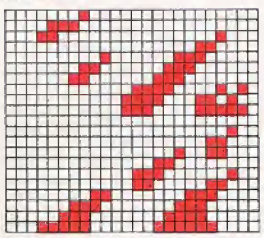
```
TO INSECTOS
  PREPARACION MOVER.INSECTOS
END

TO PREPARACION
  DRAW FULLSCREEN TELL 0 HT PU SETXY
  (−100)(−100) TRI 200 POSICION 1 (−100)
  (−100) POSICION 2 0 73 POSICION 3 100
  (−100)
END

TO TRI :LADO
  PD REPEAT 3 [FD :LADO RT 120] PU
END

TO POSICION :NUM :X :Y
  TELL :NUM SETSHAPE 3 PU SETXY :X :Y PD ST
END

TO MOVER.INSECTOS
  SEGUIR 1 2 SEGUIR2 3 SEGUIR 3 1
  MOVER.INSECTOS
END

TO SEGUIR :A :B
  TELL :B MAKE"X XCOR MAKE "Y YCOR TELL :A
  SETH TOWARDS :X :Y FD 10
END
```

```
  TELL :B
  MAKE"X2 XCOR
  MAKE"Y2 YCOR
  OUTPUT SQRT((:X1 − :X2)*(:X1 −:X2)+(:Y1
  − :Y2)*(:Y1 −:Y2))
END

TO CIUDAD.DESTRUIDA
  TELL :CIUDAD
MAKE"X XCOR
  MAKE"Y YCOR
  FLASH :X :Y :NARANJA
  TELL :CIUDAD1
  MAKE"X2 XCOR
  MAKE"Y2 YCOR
  FLASH1 :X2 :Y2 :NARANJA
  ESCONDER.SPRITE
  SPLITSCREEN
  REPEAT 3 [PRINT"]
  PRINT [DESTRUIDA LA CIUDAD!]
END

TO FLASH :X :Y :COLOR
  TELL :LLAMA
  PENCOLOR :COLOR
  SETXY :X :Y
  ST
  REPEAT 6 [SMALLX SMALLY ESPERAR BIGX
  BIGY ESPERAR]
END

TO FLASH1 :X2 :Y2 :COLOR
  TELL :LLAMA1 PENCOLOR :COLOR
SETXY :X2 :Y2
  ST REPEAT 6 [SMALLX SMALLY ESPERAR BIGX
    BIGY ESPERAR]
END

TO MOVIMIENTO.CABALLERO
  TELL :DRAGON
  MAKE"X XCOR
  MAKE"Y YCOR
  TELL :CABALLERO
  SETH TOWARDS :X :Y
  FD 10
END

TO ACERTADO?
  TELL :DRAGON
  IF TS? THEN OUTPUT "TRUE
  OUTPUT "FALSE
END

TO DRAGON.DESTRUIDO
  TELL :DRAGON
  MAKE "X XCOR
  MAKE "Y YCOR
  FLASH :X :Y :NEGRO
  ESCONDER.SPRITE
  SPLITSCREEN
  REPEAT 3[PRINT"]
  (PRINT[DRAGON MUERTO A UNA DISTANCIA
    DE] DISTANCIA :DRAGON :CIUDAD)
END

TO ESCONDER.SPRITE
  TELL :LLAMA HT
  TELL :LLAMA1 HT
  TELL :CIUDAD HT
  TELL :CIUDAD1 HT
END

TO ESPERAR
  REPEAT 100 []
END
```

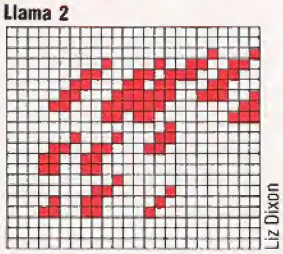
Editor de sprites



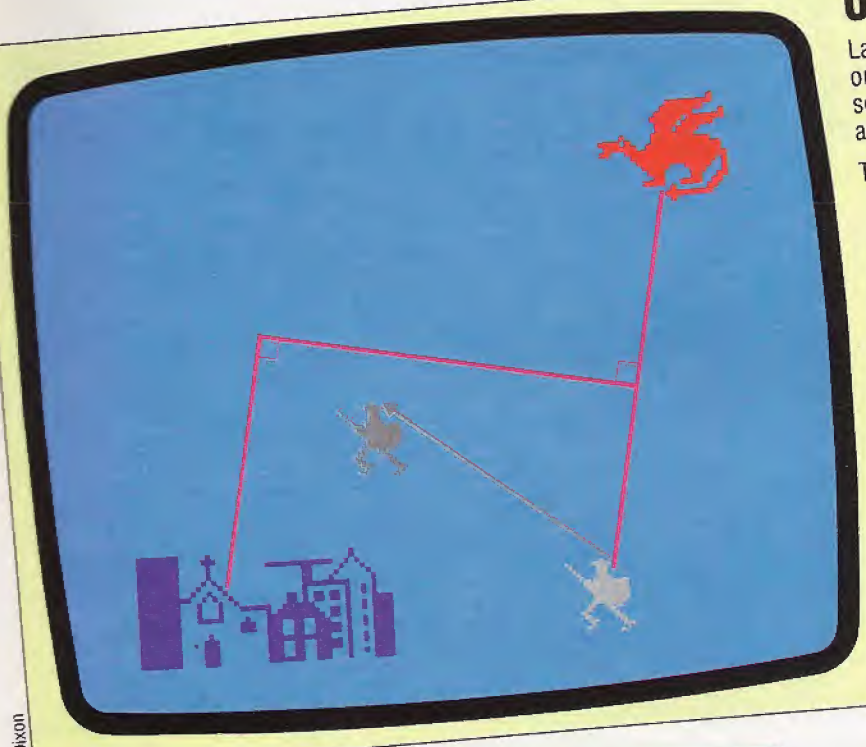
Llama 1



Llama 2



Liz Dixon



Liz Dixon

Una estrategia mejor

La mejor estrategia (para ambas partes) es orientarse hacia el lugar más próximo a la ciudad sobre la bisectriz perpendicular de la línea que une al caballero y al dragón (véase diagrama):

```
TO MOVER.CABALLERO
  TELL :DRAGON MAKE "DX XCOR MAKE"DY
  YCOR TELL :CABALLERO MAKE"KX XCOR
  MAKE "KY YCOR TELL :CIUDAD MAKE"CX
  XCOR MAKE "CY YCOR
  MAKE"SX(:DX+:KX)/2
  MAKE"SY(:DY+:KY)/2
  MAKE"VX(:DY-:KY)
  MAKE"VY(:KX-:DX)
  MAKE"FACT(:VX*(:CD-:SX)+:VY*
  (:CY-:SY))/((:VX*:VX)*(:VX*:VX)+
  (:VY*:VY)*(:VY*:VY))
  MAKE"X :SX+:FACT*:VX
  MAKE"Y :SY+:FACT*:VY
  TELL :CABALLERO SETH TOWARDS :X :Y
  FD 10
END
```

Movimiento acertado

Un movimiento mejorado para el caballero, en el cual éste se orienta para interceptar al dragón:

```
TO MOVIMIENTO.CABALLERO
  TELL :DRAGON MAKE"X XCOR MAKE "Y
  YCOR
  TELL :CIUDAD
  MAKE"DIRIGIENDOSE TOWARDS :X :Y
  TELL :CABALLERO SETH 270+:DIRIGIENDOSE
  IF XCOR< :X THEN LEFT 180
  FD 10
END
```

Abreviaturas

BACKGROUND	BG
PENCOLOR	PC

Control por teclado

Control por teclado del dragón:

```
TO MOVIMIENTO.DRAGON
  TELL :DRAGON MOVER TECLALEIDA FD 10
END

TO MOVER :DIR
  IF :DIR="W THEN SETH 0
  IF :DIR="S THEN SETH 90
  IF :DIR="Z THEN SETH 180
  IF :DIR="A THEN SETH 270
END

TO TECLALEIDA
  IF RC? THEN OUTPUT READCHARACTER
  OUTPUT""
END
```

Complementos al LOGO

El LOGO Spectrum y el LOGO Apple no incorporan gráficos sprite.

Los usuarios de Atari deben tomar nota de estas diferencias:

1) TS? no existe. Omita la línea IF ACERTADO? etc. de JUGAR e inserte la siguiente línea como la última de PREPARAR.PANTALLA:

```
  WHEN TOUCHING :DRAGON :CABALLERO
  [DRAGON.DESTRUIDO STOP]
```

2) No hay equivalentes para BIGX, BIGY, SMALLX y SMALLY. Omítalas.

3) Para BACKGROUND utilice SETBG, y para PENCOLOR emplee SETPC. Los códigos de color son diferentes.

4) Lo más sorprendente es que TOWARDS no existe en el LOGO Atari (sí se incluye en las versiones LCSI para el Apple y el Spectrum). Por lo tanto, reemplace las líneas:

```
  SETH TOWARDS :X :Y
  FD 10
```

de SEGUIR y MOVIMIENTO.CABALLERO por:

```
  MAKE"FRAC 10/(SQRT((XCOR-:X)*(XCOR
  -:X)+(YCOR-:Y)*(YCOR-:Y)))
  SETPOS LIST (XCOR+(:X-XCOR)*:FRAC)
  (YCOR+(:Y-YCOR)*:FRAC)
```

En suspenso

Vamos a estudiar detalladamente el mecanismo de las interrupciones al que repetidamente nos venimos refiriendo

Una conocida aplicación de las interrupciones es la de espera de la pulsación de una tecla del teclado. Si un programa accediera directamente a éste —que suele hacerse a través del sistema operativo— para obtener el siguiente carácter de entrada, entonces cualquier tecla pulsada cuando el programa estuviera haciendo otra cosa quedaría sin efecto. Incluso cuando el procesador se aplica de lleno a procesar la entrada del teclado, es posible que pierda algún carácter, en especial el que siga al carácter que requiera un proceso extra, como es el retorno de carro.

El teclado encuentra la solución interrumpiendo al procesador en cuanto se oprime una tecla, de modo que éste detiene lo que está haciendo y ejecuta la "rutina de atención de interrupción". Tal rutina toma el carácter que acaba de introducirse y lo coloca en una sección de memoria reservada con el nombre de *buffer del teclado*. Seguidamente el procesador puede volver a lo que estaba haciendo como si nada hubiera ocurrido.

Siempre que se llama a la rutina de entrada de teclado del sistema operativo, ésta no inspecciona directamente el teclado sino que toma el carácter siguiente del buffer (si el buffer está vacío esperará a que aparezca un carácter). Este mecanismo permite al usuario "adelantarse" en el tecleado respecto a lo que aparece en pantalla, y a la vez asegura que no se pierda ningún carácter.

Hay, sin embargo, dos problemas posibles. El usuario puede teclear tan rápido que el buffer se llene más aprisa de lo que el programa tarda en tratar las entradas, produciendo su desbordamiento (*overflow*). Esto se resuelve con un compromiso en el tamaño del buffer, de tal forma que no se produzcan desbordamientos y no se derroche un excesivo espacio de memoria, que tan valioso puede resultar. El segundo problema surge con aquellos usuarios que se sienten incómodos cuando un carácter no aparece en la pantalla inmediatamente después de haber oprimido una tecla. Puede que entonces continúen oprimiéndola y, de resultas, generen docenas de caracteres que van a parar al buffer, hasta desbordarlo de nuevo. Un problema que se resuelve familiarizándose con el ordenador.

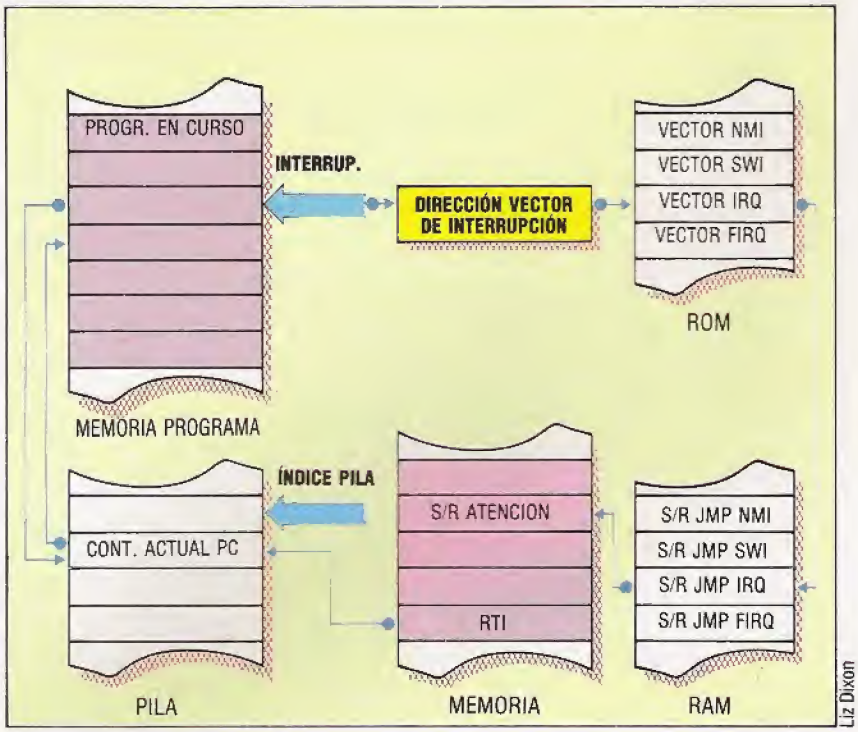
Otra aplicación útil de las interrupciones sucede cuando hay una salida hacia la impresora, que es con frecuencia una de las operaciones ejecutadas por el micro que más tiempo requieren. Durante la impresión, puede que se le exija al procesador trabajar durante 100 microsegundos mientras envía un carácter a la impresora y esperar después miles de microsegundos hasta que la impresora procese ese carácter. Para obviar la dificultad se recurre a un sistema de *spooling* (de SPOOL: *Simultaneous Peripheral Operation On Line*: operación simultánea de periféricos en línea). Éste sitúa en una cola los archivos a imprimir, y parte del primer archivo que está en la cola es almacenada en otra área del buffer de la memoria. La puerta que está al servicio de la impresora interrumpirá al procesador en cuanto la impresora esté preparada para la recepción de otro carácter. La rutina de atención de interrupción enviará entonces el siguiente carácter desde el buffer, o (si el buffer está vacío) cargará la siguiente sección del fichero que encabeza la cola, llevándola al buffer. De este modo, la impresora puede continuar su trabajo entre bastidores, mientras el procesador queda libre para atender cualquier otra cosa.

Tipos de interrupción

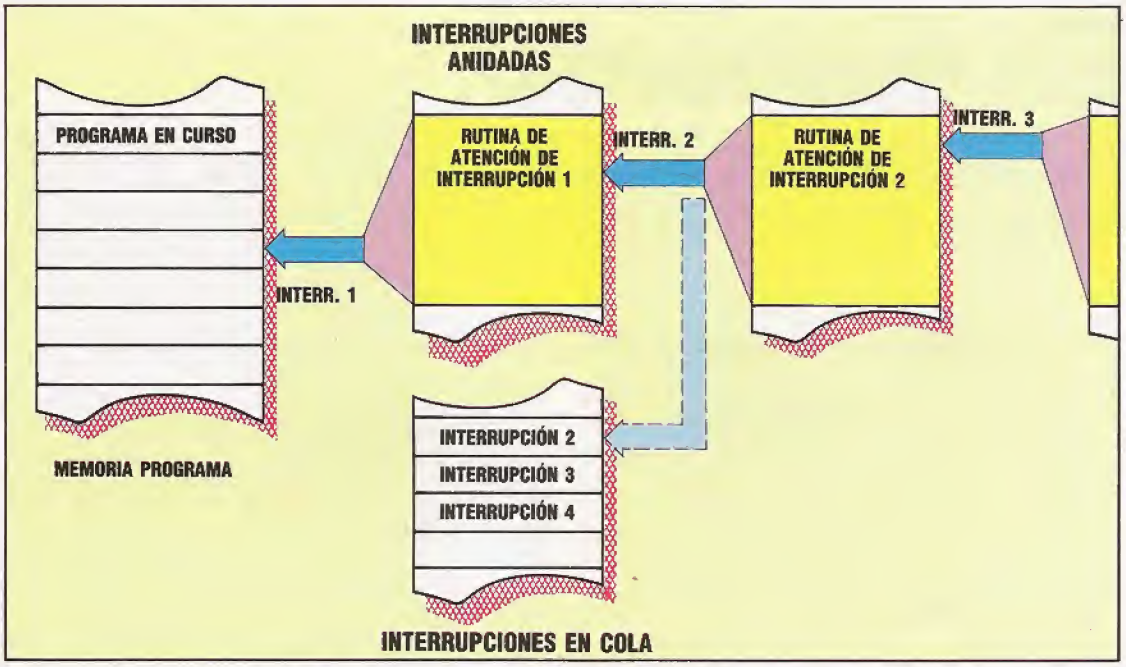
Existen operaciones realizadas por el procesador (el acceso a discos es una de ellas) en las que una interrupción puede causar pérdida de datos o cualquier otro incidente. Debe haber, por tanto, un mecanismo para "enmascarar" interrupciones de modo que el procesador pase por alto las que puedan ocurrir durante una operación particularmente delicada. En este caso, es aconsejable anotar que ha habido una interrupción y que será tratada más adelante.

Por otro lado, si estamos manejando una interface de discos llevada por medio de interrupciones, éstas deben tener prioridad y no han de enmascararse bajo ningún pretexto: es lo que ha motivado el concepto de interrupción *no enmascarable*. Tal

Interrupción número uno
Al ocurrir una interrupción el procesador concluye la ejecución de la instrucción en curso y guarda en la pila el contenido actual del contador de programa. La dirección vector de la interrupción adecuada se carga en dicho contador, pasándose el control a dicha dirección, que suele encontrarse en la ROM. Esta dirección señala a su vez a otra de la RAM, donde la instrucción JMP dirige el control a la rutina de atención de interrupción requerida. La cual concluye con una instrucción RTI para devolver el control al programa principal por medio de la dirección de retorno guardada en la pila. Dado que la instrucción JMP está almacenada en la RAM, el programador puede encontrarla y alterarla de tal modo que el control pueda ser pasado primero a una rutina de usuario con finalidad específica y después a la rutina de atención habitual



Interrupciones interrumpidas
Si durante una interrupción tuviera lugar otra interrupción, el procesador puede optar por "anidar" las interrupciones: cada vez que se da una interrupción, el PC es salvado en una pila para ocuparse de la nueva interrupción inmediatamente y devolver después el control a la dirección que se salvó en la pila. Los límites de estos anidamientos dependen de la capacidad de las pilas y de la posibilidad de los dispositivos generadores de interrupciones para soportar demoras en el proceso de sus interrupciones.
Otra opción del procesador consiste en apilar los detalles de toda interrupción en una cola de interrupciones. Concluida la primera interrupción, el procesador inspecciona la cola y procesa la interrupción que encuentre allí por orden hasta que la cola se vacía pasando después el control al programa



interrupción puede proceder de un circuito que detecte una caída en la tensión de la corriente eléctrica: su rutina de atención debe comenzar inmediatamente a salvar el estado de la tarea actual mientras quede electricidad.

Si las interrupciones pueden proceder de más de una fuente, tendremos que considerar la idea de interrupciones *anidadas*. Si sucede una interrupción mientras el procesador está atendiendo otra interrupción, hay dos posibles estrategias para su tratamiento. Primero, que se deseche la nueva interrupción mientras no se complete la que está en curso. Segundo, que se jerarquicen las interrupciones según un criterio de urgencia, de tal modo que una interrupción de máxima prioridad pueda detener el tratamiento de otra con menor prioridad. En este caso el sistema operativo ha de saber tratar el anidamiento de rutinas de atención de interrupciones.

Interrupciones de software

La instrucción SWI, mencionada brevemente en la página 1057, puede ser empleada en un programa para volver convenientemente al sistema operativo mediante la generación de su propia interrupción, llamada *interrupción de software* (para distinguirla de las que genera el hardware y de las que hasta ahora hemos estado hablando). Pero también podemos utilizar instrucciones SWI para que actúen como puntos de discontinuidad o ruptura en un programa en lenguaje máquina como ayuda a la depuración de errores; esta facilidad la procuran muchos monitores de código máquina basados en ROM, así como los paquetes de depuración. El usuario escoge puntos en la codificación donde hará una pausa la ejecución del programa, y las instrucciones colocadas en estas posiciones son reemplazadas por las SWI. Cuando se ejecuta el programa, la rutina de atención de interrupciones permite al programador inspeccionar y, si es el caso, alterar el contenido de los registros y las posiciones de la memoria, y ver lo que el programa está haciendo exactamente. Reanudada la ejecución, el monitor-depurador sustituye la instrucción indicada por el punto de ruptura SWI para que continúe el programa desde ese mismo punto.

El 6809 tiene tres mecanismos de interrupción independientes: IRQ (Interrupción ReQuerida), FIRQ (Primera —*First*— Interrupción ReQuerida) y NMI (Interrupción No enMascarable). Las tres son activadas por una señal apropiada que se recibe en tres patillas del chip del procesador. La barra sobre el nombre (p. ej., en $\overline{\text{IRQ}}$) sirve para indicar que son activadas por una señal 0 en el procesador, en lugar de una señal 1. Estas tres patillas se conectan al bus principal para que los chips periféricos tales como el 6820 y el 6850 puedan conectar sus patillas de salida de interrupción requerida a las mismas líneas del bus. Cuando se programan los chips, las interrupciones pueden ser permitidas y el envío de las señales apropiadas puede ser automático.

Hay también tres interrupciones de software causadas por las instrucciones SWI, SW12 y SW13.

Cuando se produce una interrupción, el control se pasa a la dirección de *vector* contenida en una posición específica en la parte superior de la memoria. Estas direcciones de vector se encuentran generalmente en la ROM, por lo que el control pasa desde allí hasta la misma dirección fijada. Pero esta dirección a su vez suele encontrarse en la RAM y contendrá una instrucción JMP, por cuyo motivo puede cambiarse el destino final hacia la rutina de atención propia del usuario. Las posiciones de memoria son éstas:

Tipo de interrupción	Vector
$\overline{\text{NMI}}$	$FFFC
SWI	$FFFA
$\overline{\text{IRQ}}$	$FFF8
$\overline{\text{FIRQ}}$	$FFF6
SW12	$FFF4
SW13	$FFF2

Es conveniente observar también que los dos bytes superiores de la memoria (el $FFFE y el $FFFF) con-

tienen el *vector de reinicialización*, la dirección a la que se transfiere el control al dar la corriente eléctrica o reinicializar el hardware; suele tratarse de la dirección de inicio del monitor de la ROM. Asimismo, los dos bytes situados en $FFF0 y $FFF1 los reserva Motorola para un posible empleo futuro.

La información sobre interrupciones se halla en tres bits del registro de código de condición (CC): el bit 4 (I), el bit 6 (F) y el bit 7 (E). Si el bit I se pone a uno se enmascara la interrupción IRQ, si el que se pone a uno es el bit F enmascaramos la FIRQ. El bit E sirve al procesador para distinguir entre IRQ o NMI y FIRQ: si E se pone a uno es que ha ocurrido una IRQ o NMI, y si se pone a cero, una FIRQ. Cuando se recibe una interrupción, ésta suele tener el mismo tratamiento que una llamada de subrutina: los contenidos de algunos o todos los registros son llevados a una pila para permitir que el control pueda retornar al mismo punto por donde iba la ejecución del programa. La rutina de servicio de interrupción acaba con una instrucción RTI (parecida a una RTS), que tiene la virtud de restaurar los registros de la pila y devolver el control al programa original.

La diferencia entre la FIRQ y las otras dos interrupciones reside en que FIRQ sólo guarda en la pila el registro contador del programa (PC) y el de código de condición (CC), resultando así mucho más rápida operativamente que las demás. La rutina de atención de interrupciones debe, sin embargo, restaurar cualquier registro empleado, lo que desaconseja este tipo de interrupción en rutinas que utilicen más de uno o dos registros. Estamos ahora en condiciones de ver dónde se emplea el bit E, dado que se emplea la misma instrucción RTI para concluir IRQ y FIRQ, aunque el procesador debe determinar qué registros han de ser restaurados de la pila. La secuencia de operaciones es la siguiente:

1) Se ejecuta la instrucción en curso.
2) El bit I se pone a uno, impidiendo más interrupciones IRQ. Si se trata de una interrupción FIRQ o NMI entonces también se pone a uno el bit F para inhibir la FIRQ. Aunque SW12 y SW13 no enmascaran otras interrupciones, la SW1 sí lo hace.
3) En caso de una FIRQ el bit E se pone a cero, de lo contrario se pone a uno.
4) El vector colocado en las posiciones apropiadas de memoria es cargado en el PC y la ejecución continúa a partir de esa dirección.

Nuestro primer programa propone otro útil empleo de las interrupciones: el mantenimiento de un reloj de tiempo real. Vamos a suponer que algún tipo de dispositivo temporizador (p. ej., un chip de propósito especial, como el temporizador 6840) se ha conectado al PIA en la posición $5000. La primera subrutina permitirá las interrupciones y establecerá en $50 un contador de 16 bits. La rutina de atención de interrupciones se limitará a incrementar el contador para que en cualquier momento la inspección de $50 proporcione el número de señales de temporizado que se han recibido y a partir de las cuales se puede calcular el tiempo si se conoce el tiempo inicial y la frecuencia de dichas señales.

El segundo programa ejemplo supone la conexión de una impresora al mismo PIA en $E000. Emplearemos un buffer de longitud indefinida en la posición $100 para guardar en él una línea de salida que será impresa por la rutina de atención. En $50 se pone un flag a cero mientras se está imprimiendo la línea, y a uno al concluir su impresión. Esto último permitirá la introducción de otra rutina para volver a llenar el buffer y de la que de momento no nos ocupamos aquí. $51 y $52 contienen un apuntador al buffer que proporciona la dirección del siguiente carácter a imprimir. La primera subrutina establece el PIA, el flag y el apuntador al buffer preparados para una nueva línea.

Programa núm. 1

```
PIACR   EQU   $E001     Registro de control del PIA
PIADR   EQU   $E000     Registro de datos del PIA
INTRP   EQU   $2000
CLK1    EQU   $50
CLK2    EQU   $51
        ORG   $1000     Subrutina que inicializa el reloj
INITCK  CLR   CLK1      Limpia posiciones del reloj
        CLR   CLK2
        LDA   #%00000101  Posibilita interrupciones del PIA
        STA   PIACR
        ANDCC #%11101111  Posibilita la IRQ
WAITCK  TST   CLK2      Espera el primer incremento
        BEQ   WAITCK
        RTS
        ORG   INTRP     Rutina de atencion de la interr.
        LDA   PIADA     Borra la interrupcion
        LDD   CLK1      Toma el contador
        ADDD  #1        Incrementa el contador
        STD   CLK1
        RT1
```

Programa núm. 2

```
PIACR   EQU   $E001     Registro de control del PIA
PIADR   EQU   $E000     Registro de datos del PIA
INTRP   EQU   $2000
CR      EQU   13        Retorno de carro
BUFFER  EQU   $100      Direccion del buffer
BUFPTR  EQU   $51       Apuntador del buffer
FLAG    EQU   $50       Flag de fin de linea
        ORG   $1000     Subrutina para la prep. global
        CLR   FLAG      Borra el flag de fin de linea
        LDX   #BUFFER   Inicializa el apuntador del buffer al
                        comienzo de este
        STX   BUFPTR
        CLR   PIACR     Registro de direccion de datos
        LDA   $FF       Prepara lineas para la salida
        STA   PIADR
        LDA   #%00000101  Posibilita interrupciones del PIA
        STA   PIACR
        ANDCC #%11101111  Posibilita la IRQ
        RTS
        ORG   INTRP     Rutina de atencion interr.
        LDX   BUFPTR    Apuntador del buffer
        LDA   ,X+       Toma del buffer el sig. car.
        STA   PIADR     Lo imprime
        LDB   PIADR     Borra la interrupcion
        STX   BUFPTR    Apuntador del buffer incrementado
        CMPA  #CR       Fue fin de linea?
        BNE   FINISH    Salto si no fue asi
        INC   FLAG      Activa el flag, si fue asi
FINISH  RTI
```

Para el Commodore

Esta vez desarrollaremos para el Commodore 64 un programa de utilidad para búsqueda de variables, tal como lo hiciéramos para el BBC Micro y el Spectrum (pp. 1144-1145)

En la versión Commodore del programa se deben abreviar muchos nombres de variables para evitar que incluyan una palabra clave de BASIC. Por ejemplo, NEWLINE (nueva línea) no se puede utilizar como nombre de una variable porque comienza con NEW, y TEXTPOINTER (apuntador de texto) tampoco se puede emplear porque incluye INT.

Los cambios en la parte inicial del programa son necesarios debido a las diferencias en cuanto a la forma en que se almacena una línea de BASIC en la memoria del ordenador. En el BBC Micro y el Spectrum, en el formato interno una línea de BASIC empieza con un número de línea de dos bytes, con el byte más significativo primero, y uno o dos bytes para la longitud de la línea. En el Commodore 64, una línea de BASIC comienza con un apuntador de dos bytes que indica el comienzo de la línea siguiente, y un número de línea de dos bytes, con el byte menos significativo primero en ambos casos.

Aún debemos saltar las líneas REM y las series entre comillas, pero no es necesario buscar ningún otro caso especial que pudiera causar confusión, como los números hexadecimales en el BBC o la forma binaria oculta de números en el Spectrum.

Utilidad de búsqueda para el C64

```
30000 INPUT "NOMBRE A BUSCAR";T$
30010 RE=143
30020 COMILLA=34
30030 NL=0
30040 APNTEXTO=2049
30050 SIGLINEA=PEEK(APNTEXTO)+256*PEEK (APNTEXTO+1)
30070 APNTEXTO=APNTEXTO+2
30080 NUMLINEA=PEEK(APNTEXTO)+256*PEEK(APNTEXTO+1)
30085 IF NUMLINEA>=30000 THEN END
30090 APNTEXTO=APNTEXTO+2
30140 IF PEEK(APNTEXTO)=NL THEN APNTEXTO=APNTEXTO+1:GOTO 30050
30150 IF PEEK(APNTEXTO)<>RE THEN GOTO 30180
30160 REM SALTAR LINEA REM
30170 APNTEXTO=SIGLINEA:GOTO 30050
30180 IF PEEK (APNTEXTO)<>COMILLA THEN GOTO 30300
30190 REM SALTAR TODO LO QUE ESTE ENTRE COMILLAS, PARARSE AL FINAL DE LA LINEA EN CASO DE COMILLA SIN
      PAREJA
30200 APNTEXTO=APNTEXTO+1
30210 IF PEEK(APNTEXTO)=NL THEN APNTEXTO=APNTEXTO+1:GOTO 30500
30220 IF PEEK(APNTEXTO)<>COMILLA THEN GOTO 30200
30230 APNTEXTO=APNTEXTO+1
30235 GOTO30140
30290 REM EL PRIMER CARACTER DEL NOMBRE DEBE SER UNA LETRA
30300 IF PEEK(APNTEXTO)>=ASC("A") AND PEEK (APNTEXTO)<=ASC("Z") THEN GOTO 30330
30310 APNTEXTO=APNTEXTO+1
30320 GOTO30140
30330 NOMBRE$=""
30340 NOMBRE$=NOMBRE$+CHR$(PEEK(APNTEXTO))
30350 APNTEXTO=APNTEXTO+1
30360 REM LETRA O DIGITO DESPUES DEL PRIMER CARACTER
30370 IF PEEK(APNTEXTO)>=ASC("A") AND PEEK(APNTEXTO)<=ASC("Z") THEN GOTO 30340
30390 IF PEEK(APNTEXTO)>=ASC("0") AND PEEK(APNTEXTO)<=ASC("9") THEN GOTO 30340
30410 REM FINAL CON $ PARA VARIABLE EN SERIE, % PARA VARIABLE ENTERA
30420 IF PEEK(APNTEXTO)=ASC("$") THEN NOMBRE$=NOMBRE$+"$":APNTEXTO=APNTEXTO+1:GOTO 30450
30430 IF PEEK(APNTEXTO)=ASC("%") THEN NOMBRE$=NOMBRE$+"%":APNTEXTO=APNTEXTO+1
30440 REM (SI MATRIZ O FUNCION
30450 IF PEEK(APNTEXTO)=ASC("(") THEN NOMBRE$=NOMBRE$+"(":APNTEXTO=APNTEXTO+1
30460 IF NOMBRE$=T$ THEN PRINT NOMBRE$;" EN LA LINEA";NUMLINEA
30470 GOTO 30140
30480 END
```

La sección del programa que realmente extrae los nombres de variables busca primero una letra del alfabeto, luego letras o dígitos, y finalmente busca un signo $ o % que indique una variable en serie o entera y un carácter (, que indicaría una función o matriz. El Commodore 64 no admite el carácter de subrayado que sí se puede incluir en los nombres de variables en el BBC Micro y el Spectrum.

Si bien el Commodore 64 puede visualizar tanto letras en mayúscula como en minúscula, la diferencia sólo existe en cuanto a la forma del carácter que aparece en la pantalla, y no en el código interno para el carácter. Por tanto, el programa sólo ha de buscar las mayúsculas en el nombre de una variable.

La versión del programa para el Commodore 64 se utiliza de la misma manera que las versiones para el BBC Micro y el Spectrum. Entre el programa de búsqueda y guárdelo (SAVE), luego cargue (LOAD) el programa en el cual se realizará la búsqueda y añádale el programa de búsqueda. Entonces ya puede buscar en el programa mediante "RUN 30000" y tecleando el nombre de la variable cuando el programa se lo solicite, terminando el nombre con "(" si desea hallar el nombre de una matriz.

Existe un procedimiento sencillo para unir dos programas guardados en el Commodore 64, siempre y cuando los números de líneas del primer programa sean todos inferiores a los números de línea del segundo programa. El método utiliza dos de los apuntadores de la página cero: TXTTAB, en las direcciones 43 y 44, que retiene la dirección donde comienza el programa en BASIC, y VARTAB, en las direcciones 45 y 46. Un programa en BASIC acaba con un byte conteniendo cero que indica el final de la última línea del programa, luego otros dos bytes cero que marcan el final del programa. La dirección en VARTAB normalmente es el byte que sigue al último de estos ceros. Para unir los dos programas, primero cargue (LOAD) el programa que tenga los números de línea inferiores, después digite:

```
PRINT PEEK(45), PEEK(46)
```

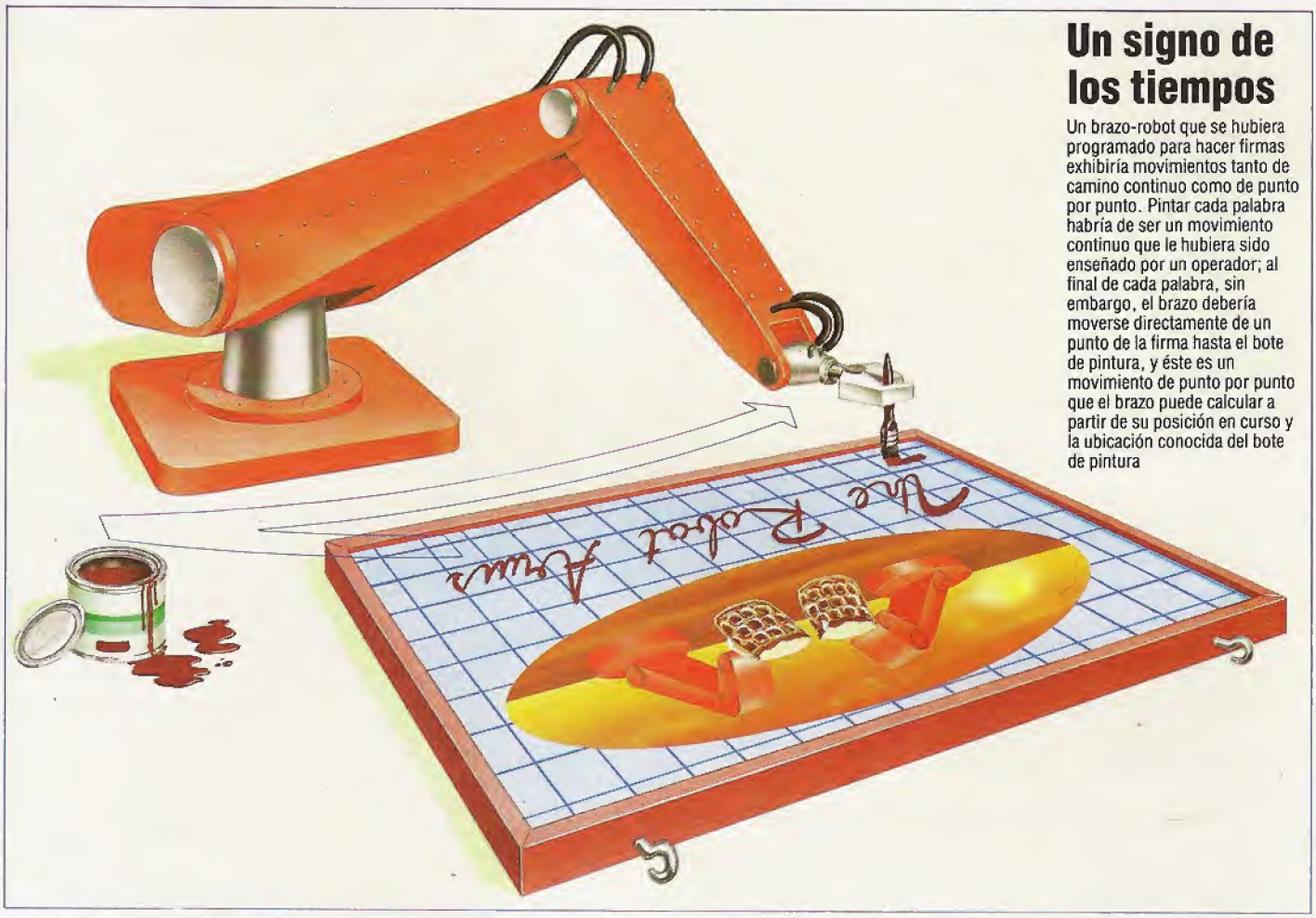
Si el primer número está comprendido entre 2 y 255, réstele 2 y coloque (POKE) el resultado en la dirección 43. Si es cero o uno, digite POKE 254 o 255 en la dirección 43 y coloque (POKE) en la dirección 44 uno menos que el resultado de PEEK(46). Entonces puede cargar el segundo programa, y finalmente debe digitar:

```
POKE 43,1: POKE 44,8
```

Esto volverá a colocar el valor normal en el apuntador de "comienzo de BASIC", y ahora los programas se unirán entre sí.

Movimientos exactos

En esta oportunidad abordaremos diversos aspectos de la programación de brazos-robot



Steve Cross

Un signo de los tiempos

Un brazo-robot que se hubiera programado para hacer firmas exhibiría movimientos tanto de camino continuo como de punto por punto. Pintar cada palabra habría de ser un movimiento continuo que le hubiera sido enseñado por un operador; al final de cada palabra, sin embargo, el brazo debería moverse directamente de un punto de la firma hasta el bote de pintura, y éste es un movimiento de punto por punto que el brazo puede calcular a partir de su posición en curso y la ubicación conocida del bote de pintura

Ya hemos visto cómo se pueden construir brazos-robot que se asemejen a una extremidad humana: poseen "esqueleto" para proporcionar una estructura y "músculos" para proporcionar energía motriz. Pero el brazo aún necesita "inteligencia" para poder llevar a cabo tareas.

La idea de un brazo inteligente a primera vista parecería absurda. Sin embargo, la forma de inteligencia que estamos considerando aquí no corresponde a aquella de alto nivel que poseen los seres humanos, sino a algo considerablemente menos complejo. Tomemos a modo de ejemplo una acción humana sencilla. Usted se encuentra sentado junto a una mesa que está vacía, a excepción de un pequeño objeto que se halla colocado a la izquierda. Su tarea consiste en trasladar este objeto desde la parte izquierda hasta la parte derecha de la mesa. Aquí intervienen dos formas de inteligencia. La primera implica la percepción tanto de la mesa como del objeto y la decisión de desplazar éste de uno a otro lado. Esto presupone un "pensamiento consciente", y está relacionado con conceptos tales como "intención" y "comportamiento orientado hacia un objetivo". La inteligencia que necesitamos considerar es muy elemental: la que se precisa para mover el brazo y la mano correctamente *después* de haber decidido la tarea a cumplir: desplazar la mano hasta la posición correcta y asegurarse de que coja el objeto y lo suelte en el momento preciso.

Entrenamiento humano

Esto parece a la vez sencillo y obvio; pero si tiene alguna duda en el sentido de que este acto sea realmente inteligente, tan sólo observe a un niño pequeño tratando de seguir la misma secuencia. El pequeño a menudo no conseguirá asir el objeto, lo desplazará hasta una posición inadecuada y es probable que parezca bastante inseguro sobre qué es lo que se requiere. El niño está intentando adquirir la inteligencia necesaria para mover sus brazos y manos en un mundo tridimensional que le es extraño. Después de aprendido esto, tales movimientos se realizarán de forma automática, sin exigir ningún pensamiento consciente, y entonces dejará de pensar en ellos como si exigieran inteligencia.

El brazo-robot se encuentra en la misma situación que el niño que empieza a andar: posee el equipo para realizar tareas, pero debe "aprender" a realizarlas de forma automática.

El método más sencillo consiste en entrenar el brazo para que efectúe labores específicas con sólo guiarlo a lo largo de una secuencia de movimientos y decirle que "la recuerde". Este procedimiento se utiliza en una gran cantidad de robots industriales. Un operador toma literalmente de la mano al robot y lo conduce a través de los pasos que debe dar. Esto tiene la enorme ventaja de que la persona que lo "instruye" no necesita saber nada sobre cómo funciona en realidad el brazo-robot: todo lo que ha de saber es la secuencia de acciones que debe seguir el brazo. Por su parte, el robot no necesita "saber" lo que está haciendo: simplemente tiene que "recordar" las acciones que debe llevar a cabo.

Métodos de entrenamiento

Hay dos clases de "entrenamiento" que se utilizan con los brazos-robot, el de punto por punto y el de camino continuo. En el entrenamiento de *punto por punto*, el operador mueve el brazo hasta una cierta posición y luego pulsa un botón para señalarle al robot que debe "recordar" esa posición. El brazo es movido hasta la siguiente posición y se vuelve a pulsar el botón. Esta secuencia prosigue hasta haber almacenado en la memoria del autómata cibernético una secuencia completa de acciones. Una vez concluida la sesión de entrenamiento, se pone al robot en modalidad de "reproducción" *(playback)* y éste entonces se mueve de un punto al otro exactamente en la manera en que se le "enseñó". En el entrenamiento de *camino continuo*, el operador conduce al robot a través de la secuencia completa y éste recuerda todas y cada una de las posiciones de la secuencia. Al reproducirla, el robot sigue la secuencia de la misma forma que antes.

Geometría de dos junturas
Al moverse desde un punto a otro, el brazo-robot de dos junturas debe girar alrededor de su pivote (ángulo R) y debe cambiar los ángulos del hombro (H) y del codo (C). Si las coordenadas cartesianas del punto actual y del punto de destino fueran (X1, Y1, Z1) y (X2, Y2, Z2), los cambios se calcularían del siguiente modo:

A1=SQR(X1^2+Y1^2+Z1^2)
A2=SQR(X2^2+Y2^2+Z2^2)

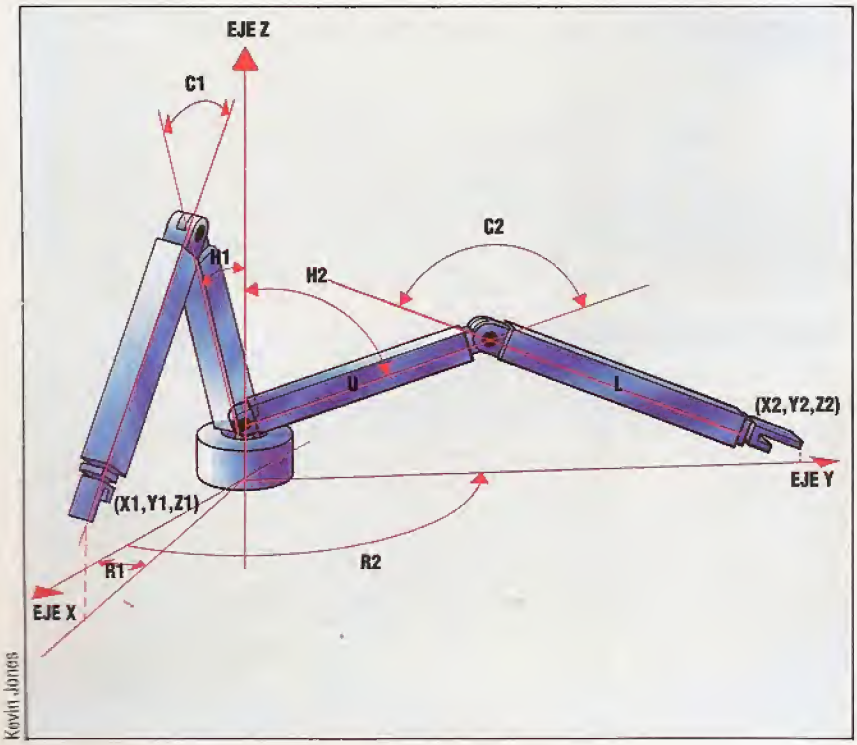
Pivote:
R1=ARCTAN(Y1/X1)
R2=ARCTAN(Y2/X2)
Cambio=(R2−R1)

Hombro:
H1=ARCCOS(Z1/A1)+ ARCCOS((A1^2+U^2− L^2)/(2*A1*U))

H2=ARCCOS(Z2/A2)+ ARCCOS((A2^2+U^2− L^2)/(2*A2*U))
Cambio=(H2−H1)

Codo:
C1=ARCCOS((U^2+L^2− A1^2)/(2*U*L))
C2=ARCCOS((U^2+L^2)− A2^2)/(2*U*L))
Cambio=(C1−C2)

donde U y L son las longitudes del brazo superior e inferior respectivamente



Kevin Jones

Retomando el ejemplo de una persona que esté sentada junto a una mesa y trasladando un objeto de uno a otro extremo de ésta, podemos utilizar el método de punto por punto para "entrenar" a un robot de modo que reproduzca esta acción. Este procedimiento se emplea con frecuencia con robots que deban llevar a cabo tareas de "asir y colocar": trasladar un objeto de un lugar a otro. Por el contrario, un robot que se utilice para atomizar pintura necesitará que se le enseñe mediante el método de camino continuo para asegurar que cubra con pintura el objeto en su totalidad, tal como lo haría una persona.

Ahora vamos a considerar cómo "recuerda" el robot la secuencia de movimientos que debe seguir. La respuesta para ello es que el robot utiliza sensores internos para grabar la posición de cada una de sus junturas durante la modalidad de entrenamiento. Esto a menudo se realiza tomando la salida de los codificadores de eje y grabando los movimientos efectuados, ya sea directamente en la memoria o bien, para un almacenamiento más permanente, en cinta o en disco. Cuando se selecciona la modalidad de reproducción, el robot puede, entonces, recuperar los datos correspondientes y convertirlos en movimiento de junturas, lo que constituye una tarea relativamente compleja.

Resulta sorprendente el hecho de que al robot le resulte más fácil "recordar" el movimiento de camino continuo: sólo tiene que seguir la ruta exacta que se le ha enseñado. No obstante, con frecuencia es necesario almacenar una cantidad de datos muy grande; a menudo se necesitan varios centenares de posiciones para definir un camino continuo, en vez de las pocas posiciones implicadas en el movimiento de punto por punto. La segunda dificultad deriva del hecho de que, si el brazo ha de seguir la secuencia uniforme y exactamente, todas sus junturas deben activarse simultáneamente. Un robot que atomice pintura tendrá que hacer un barrido del brazo a lo largo de los tres ejes, al tiempo que maniobra sus tres junturas de muñeca para posicionar el atomizador de la forma correcta. Esto significa que el ordenador que controle al robot debe trabajar muy rápido con el fin de manipular cada juntura por turno sin ninguna demora apreciable; por otra parte, el robot puede utilizar hasta seis procesadores separados, dirigiendo cada uno de ellos el movimiento de una juntura, para obtener un movimiento perfectamente simultáneo.

Movimiento calculado

Un robot de punto por punto tiene ante sí una tarea más ardua, porque si bien sabe hacia dónde moverse, no se le ha "dicho" cómo llegar hasta allí. Podría, simplemente, mover cada juntura hasta colocarla en la posición requerida, pero esto supondría una pérdida tanto de tiempo como de energía. Sería mucho mejor que el robot calculara una ruta directa para su mano desde un punto a otro; entonces podría realizar el movimiento requerido en una barrida, tal como lo hace una persona. Pero, nuevamente, los cálculos necesarios para hacer esto son complejos, dado que se deben utilizar coordenadas cartesianas para mover la mano en una línea recta entre dos puntos definidos, mientras las posiciones propias del brazo se definen según un sistema de coordenadas totalmente diferente. De modo que el robot debe ser capaz de resolver algunos in-

trincados problemas geométricos con el fin de trabajar eficazmente. Y, en el caso de robots industriales que deben trasladar a gran distancia objetos que pesan varios cientos de kilos, puede ser considerable el ahorro en tiempo y energía que supone la elección de la mejor ruta.

Otro problema que se le plantea al robot de punto por punto es la dinámica del brazo. Si usted mueve un brazo para coger un objeto, comprobará que se acelera lentamente al alejarse de su posición original hasta alcanzar una velocidad máxima, luego desacelera hasta llegar a una suave detención en su posición final. Las ventajas que ofrece un brazo-robot que hace esto son considerables. Muchos de tales brazos se mueven a una velocidad constante, acelerando casi inmediatamente hasta una velocidad máxima y deteniéndose en seco al final de la secuencia de movimientos. Esto supone un esfuerzo del brazo y exige más energía que otro que acelere y desacelere suavemente. Asimismo, significa que el brazo quizá no se mueva tan rápidamente como lo haría en el caso contrario y, si al final de la secuencia se requiere que el robot coja un objeto delicado, incluso un imperceptible desplazamiento del objeto podría provocar que el brazo golpeara contra él con una fuerza considerable. De modo que es preciso calcular una velocidad óptima para el robot además de un camino ideal a seguir.

Coreografía de robots

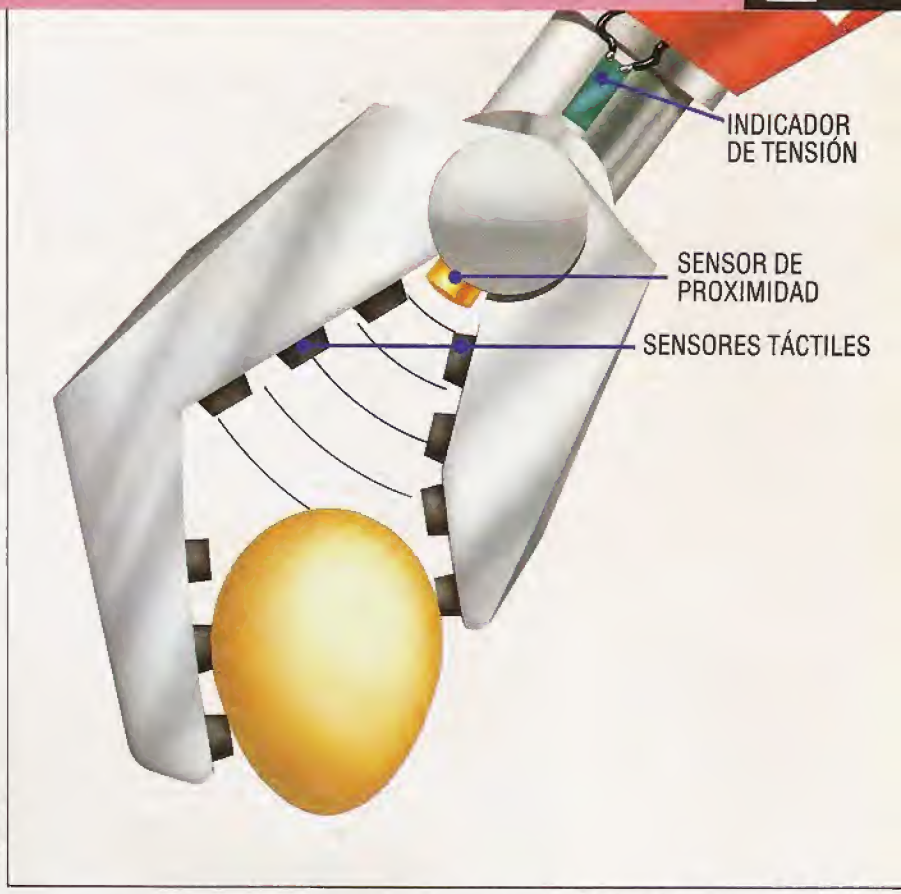
Incluso después de haber programado un brazo para que siga una serie precisa de movimientos, en el momento de la reproducción puede resultar que estos movimientos no respondan exactamente a los requerimientos. Esto puede deberse a error humano, a que la naturaleza de la tarea haya cambiado ligeramente o bien a que, si el robot está trabajando en conjunción con otros brazos, éstos sigan su propio camino y choquen entre sí. (Evitar este último problema es lo que se conoce como *coreografía de robots*.) De manera que se requiere un método para editar la secuencia. Esto se puede conseguir almacenando los movimientos como una lista encadenada, en la cual cada posición del brazo está almacenada junto con la dirección en donde se encuentra la posición siguiente. En la sesión de entrenamiento inicial, esta dirección será la de la siguiente posición de la lista. Si se necesitara editar la secuencia, se podría desplazar el brazo hasta la posición en la cual fuera necesario realizar las correcciones, detenerlo e insertar una nueva secuencia.

Otro método común para hacer que los brazos se muevan inteligentemente consiste en utilizar una serie de instrucciones programadas almacenadas en el ordenador. Típicamente, cada robot posee su propio método de programación y emplea un "lenguaje" de programación diferente para controlar el movimiento, pero en general se requiere un lenguaje que le permita al programador utilizar instrucciones tipo LOGO para especificar el movimiento en tres dimensiones, con instrucciones adicionales para movimiento de muñeca y efector final, tales como "asir" o "depositar".

El problema es similar al de entrenar un robot de punto por punto y es necesario tener en cuenta muchos factores. Por ejemplo, si se ha de mover un brazo-robot 10 unidades hacia adelante, el método más obvio sería alterar la juntura del hombro de modo que el brazo pudiera llegar más adelante. Sin embargo, esto haría que el brazo se moviera hacia arriba en un arco y, por lo tanto, esto se habría de corregir mediante un movimiento descendente en la juntura del codo. A partir de este ejemplo se puede ver que la instrucción para un solo movimiento simple se debe traducir en dos juegos distintos de instrucciones, trabajando en dos junturas separadas.

Se puede plantear otro problema cuando se requiere que el robot recoja un objeto. Por muy bien posicionado que esté el brazo, es difícil asegurar que esté exactamente en el lugar correcto para coger el objeto, en especial si éste tiene forma asimétrica. Por consiguiente, se necesita una mano "inteligente"; ésta debe percibir la presencia o la ausencia del objeto, la distancia del mismo respecto a la mano y la fuerza ejercida por ésta cuando intenta recogerlo. Estos problemas se pueden abordar equipando a la mano con una gama de sensores de proximidad, táctiles y de fuerza, que proporcionen la realimentación que capacite al ordenador controlador para efectuar cualquiera modificación necesaria.

Si se consideran todos estos problemas, podemos ver que es posible construir un brazo-robot que muestre un nivel de "inteligencia" relativamente elevado. Sin embargo, hasta el momento no se ha diseñado ningún brazo que, pongamos por caso, lance una pelota de cricket con precisión. Ello se debe a que la inteligencia del brazo por sí sola no es suficiente. El robot debe conocer, asimismo, la posición del bateador, la fuerza y la dirección del viento y un sinnúmero de otras condiciones variables. Luego deberá ser capaz de calcular las ecuaciones, complejas, que implica enviar un proyectil por el aire. Para dichas tareas se requiere mucho más que un mero brazo inteligente.



Steve Cross

Suavemente
La prueba de coger un huevo es decisiva para juzgar los sensores del brazo-robot y los mecanismos de control de realimentación. El sensor de proximidad de la uña debe comprobar que el huevo esté lo suficientemente cerca como para cogerlo, luego los dedos pueden empezar a acercarse hasta que los sensores táctiles indiquen el contacto con el huevo. La salida de los sensores táctiles se debe entonces comparar con la del sensor de proximidad a medida que los dedos se cierran y el brazo comienza a levantarse. Una disminución repentina de la proximidad indica que el huevo se está escurriendo, de modo que los dedos deben apretar hasta llegar a un límite de fuerza de agarre preestablecido o hasta que una súbita disminución de la fuerza de agarre indique que la cáscara de huevo se está deformando antes de romperse

Espacio para maniobrar

En la programación de utilidades complejas es inevitable recurrir al uso del lenguaje máquina

Con el fin de operar nuestro anterior programa de búsqueda de variables lo mezclamos con el programa en el cual se deseaba efectuar la búsqueda. Con este método, la única información relativa al sistema operativo que teníamos que proporcionar era la dirección de comienzo del programa en BASIC; el fin del programa en el cual se realizaba la búsqueda se hallaba al encontrar el número de línea más bajo del programa de utilidad.

La utilidad que estamos creando es un programa para sustituir variables. Éste es un programa muy útil para tener en el archivo. Si a lo largo de todo un programa usted hubiera empleado un nombre de variable, descubriendo después que el mismo no es legal, imagínese cuánto tiempo le ahorraría una utilidad de esta clase. Igualmente, puede ser que se haya escrito un programa y que ahora vaya a ser utilizado por otra persona, a quien tal vez no le resulte fácil descifrar los nombres de las variables. Aquí le explicamos la teoría necesaria para el código máquina y en el próximo capítulo publicaremos los listados.

Espacio de memoria

En este ejercicio necesitamos colocar el programa de utilidad en una sección de memoria distinta de aquella en la que está trabajando el programa. Asimismo, debemos hallar un método diferente de localizar el final del programa en BASIC, y un medio para acomodar en el ordenador dos programas en este lenguaje al mismo tiempo.

Los tres ordenadores que estamos considerando (BBC Micro, Commodore 64 y Sinclair Spectrum) utilizan un conjunto de apuntadores para indicar al sistema operativo y al intérprete de BASIC dónde encontrar los programas en BASIC, las variables, etc. (véase p. 536). Lamentablemente, los detalles son distintos en las tres máquinas.

En el BBC Micro hay cuatro apuntadores importantes: PAGE y TOP, que retienen la dirección de comienzo y de final del programa en BASIC; LOMEM, que retiene la dirección de comienzo de las variables de BASIC, e HIMEM, que retiene la dirección del final del área para BASIC. Estos cuatro apuntadores se almacenan como variables de BASIC incorporadas, y podemos leer o alterar sus valores mediante sencillas sentencias en este lenguaje. Si tenemos en la memoria un programa en BASIC y deseamos agregar otro, cambiamos PAGE a un valor superior a TOP (utilizando la instrucción OLD para reinicializar TOP y LOMEM) y entonces podemos agregar el nuevo programa sin afectar al programa original. Pasamos de un programa a otro dándole nuevos valores tanto a PAGE como a HIMEM y utilizando OLD.

Una vez que tenemos en ejecución el programa de utilidad, los valores de los apuntadores se refieren a este programa; para permitir que la utilidad encuentre el comienzo y el final del programa sobre el cual está trabajando, hemos de copiar los valores originales en un área de la memoria que no se altere cuando cambiemos los programas. Otro procedimiento para hallar el final de un programa es utilizar el indicador final que coloca el intérprete de BASIC. Éste es simplemente un byte que retiene un valor de 128 o más, inmediatamente después del carácter de retorno de carro del final de la última línea del programa. Este byte, y el que le siga, se interpretarán como los bytes HI y LO del número de línea siguiente. Dado que el byte HI de este número es 128 o más, éste dará un número de línea mayor o igual a 32768 (256×128). Como el máximo número de línea válido es 32767, podemos estar seguros de haber hallado el indicador de final del programa y no tan sólo otro número de línea.

El Commodore 64 utiliza siete apuntadores, almacenados en la memoria de página cero, para indicar diversas partes del área para programas en BASIC. TXTTAB, en las direcciones 43 y 44, apunta al comienzo del programa en BASIC; VARTAB, ARYTAB, STREND, FRETOP y FRESPC, de la dirección 45 a la 54, apuntan a diversas secciones de la tabla de variables, y MEMSIZ, en las direcciones 55 y 56, apunta al final del área de BASIC. Se pueden cambiar estos apuntadores con el fin de crear un área separada en el cual ejecutar un programa en BASIC mediante el empleo de la instrucción POKE. Sin embargo, se recomienda un breve programa en lenguaje máquina, porque es más directo y reduce considerablemente las probabilidades de "colgar" el ordenador por algún error de digitación.

En el Commodore 64 el final de un programa en BASIC se indica mediante dos bytes que contienen ceros inmediatamente después del byte cero que indica el final de la última línea. Siguiendo la cadena de apuntadores del principio de cada línea del programa hasta hallar un apuntador con cero, obtendremos el final del programa.

Para el Spectrum

La creación de esta utilidad es algo más complicada en el Spectrum. En vez de un área para el programa en BASIC, hay un único bloque de memoria que no sólo incluye el programa y las variables en BASIC, sino también todas las áreas de trabajo que utilizan el sistema operativo y el intérprete de BASIC. Con este trazado de la memoria es difícil, si no imposible, tener dos programas en BASIC en el área operativa principal, de modo que haremos una copia de

nuestro programa por encima de RAMTOP y trabajaremos allí. Ello aún nos deja sin resolver el problema de recuperar el programa e instalarlo en el área principal de programas después de haberlo alterado, y necesitaremos un programa en código máquina para que haga esto por nosotros.

El manual del Spectrum proporciona muchísima información sobre la forma en que se almacena un programa en BASIC y para qué se utilizan las diversas áreas de la memoria. Sin embargo, debido a la gran cantidad de secciones diferentes del área operativa, y a la forma en que se pueden desplazar estas áreas, resulta difícil escribir programas de utilidad sin emplear subrutinas en código máquina tomadas de la ROM. Si desea hacer en el Spectrum algo de programación de utilidades seria, una valiosa obra de referencia es *The complete Spectrum ROM disassembly*, de Ian Logan y Frank O'Hara. En dicha obra se explica cómo funcionan todas las rutinas de ROM.

Experimentando con BASIC

Usted puede intentar alterar el contenido de un programa durante su ejecución, pero debe guardar el programa primero, porque el resultado más común es una caída del sistema. Utilice el programa Monitor (véase p. 598), que permite inspeccionar y modificar el contenido de la memoria. Ésta se puede inspeccionar y alterar a sí misma siguiendo sus instrucciones. Inserte algunas líneas REM extras al principio del programa y pruebe en ellas estas sugerencias:

- Hallar el comienzo del área para textos en BASIC (véase p. 538) e inspeccionar el programa Monitor en la memoria hasta identificar líneas de programa.
- Cambiar los valores de los bytes después de un distintivo REM, salir luego del programa y listar la línea modificada.
- Tratar de colocar un valor mayor que 127 en una línea REM; salir y listar: quizá se sorprenda.
- Alterar los bytes de número de línea de una línea; esto produce resultados imprevisibles, en especial si el nuevo número no sigue la secuencia de sus vecinos.
- Se pueden alterar los bytes de longitud de línea, pero debe insertarse un nuevo indicador de final de línea en el byte indicado.
- En el Commodore 64 se pueden cambiar los bytes de dirección de enlace: intente sustituir la dirección de enlace de una línea por la dirección de la siguiente, y después liste el programa.
- Si es más ambicioso, consulte su manual y explore el área de almacenamiento de las variables. Ésta suele empezar en el mapa de memoria allí donde termina el área para textos de BASIC. Hay hasta seis tipos distintos de variables, cada uno de ellos con su propio formato de almacenamiento: variables numéricas, matrices numéricas, variables de enteros, matrices de enteros, variables en serie y matrices en serie. Los formatos de las variables en serie y de enteros son los más simples, siendo esencialmente representaciones directas de los datos y el nombre de la variable; los datos de las matrices numéricas son los más complicados.
- Puede tratar de modificar los valores de los distintivos en las líneas del programa: esto modificará la palabra que representa una instrucción.

FE DE ERRATAS

En la página 598, en *Complementos al BASIC* del BBC y el Commodore:

- En la línea 1150 hay dos asignaciones sucesivas a C$(3); cambie la segunda asignación por

  C$(4)="Q"
- En la línea 6600, cambie Z=1 por

  Z=2
- La línea 200 de los complementos para el BBC debe ser

  200 CLS

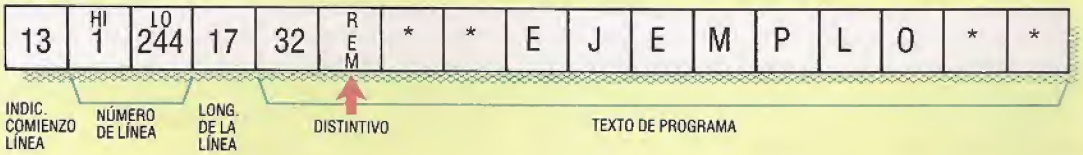
como en la versión para el Spectrum

Almacenamiento en BASIC

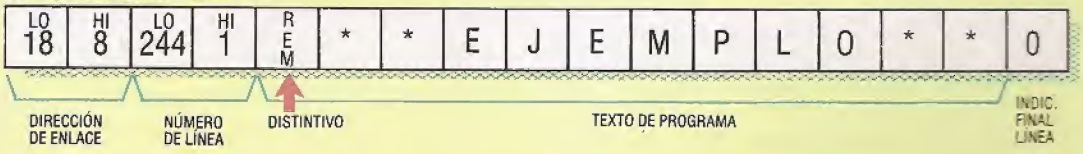
En el área para programas en BASIC, la mayoría de los micros siguen el mismo formato de almacenamiento. Cada línea de programa comienza con los datos de la línea: el número en forma de dos bytes y alguna información sobre la longitud. El texto del programa se almacena más o menos sin alteraciones, si bien las palabras clave se reemplazan por códigos de un byte denominados *distintivos*

500 REM **EJEMPLO**

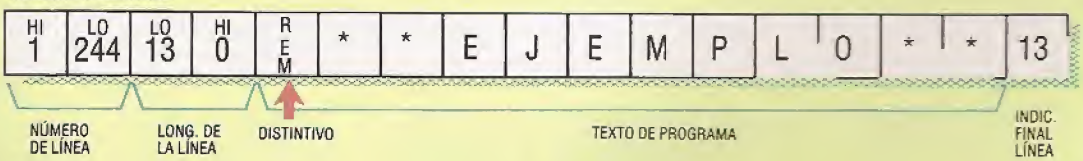
VERSIÓN BBC



VERSIÓN COMMODORE 64



VERSIÓN SPECTRUM



Este formato 13 (código ASCII para [RETURN]) es un indicador de comienzo de línea; lo más común es que esté colocado al final de la línea. La longitud de ésta sólo ocupa un único byte. Se ha almacenado el espacio que está directamente después del número de línea

Los bytes de dirección de enlace contienen la dirección en dos bytes del primer byte de la siguiente línea del programa. El área para texto comienza en la dirección 2049, y esta línea tiene 17 bytes de longitud, de modo que la dirección de comienzo de la línea siguiente es 2066

La longitud de la línea ocupa dos bytes, ¡de modo que cada línea del programa podría tener 65 535 caracteres de largo! Aquí la longitud de línea es 13: los bytes de la línea incluyendo el byte de final de línea, pero sin contar los bytes de información sobre línea

Liz Dixon

De tortugas y diablos

En este capítulo de nuestro curso de LOGO nos corresponde analizar las excepcionales facilidades para manipulación de sprites que posee el LOGO Atari

Las facilidades del LOGO Atari para la manipulación de sprites son, en realidad, notables. El usuario puede disponer de una amplia gama de colores entre los cuales escoger, además de contar con una novedosa facilidad para detectar eventos.

El LOGO Atari tiene cuatro sprites, que se numeran del 0 al 3, con el 0 como la tortuga "por defecto". El manual Atari habla de éstos como tortugas en vez de como sprites, de modo que nosotros también los llamaremos tortugas.

TELL 1 convierte a la tortuga 1 en la tortuga *en curso*; en otras palabras, la tortuga 1 obedecerá cualquier instrucción que se le dé. Pruebe con:

```
TELL 1
FD 40
RT 90
TELL 2
BK 40
RT 90
TELL 3
RT 90
```

Es posible, no obstante, tener más de una tortuga en curso. Pruebe con:

```
TELL [1 2 3]
FD 50
```

Ahora estas tres tortugas obedecerán las instrucciones que se les ha dado.

Las cuatro tortugas tienen la forma de tortuga rotatoria, la forma 0, hasta que se defina otra. Utilizando el editor se pueden definir hasta 15 formas distintas, y asignárselas después a las tortugas. Estas formas definidas por el usuario no rotan cuando la tortuga gira.

Digitando EDSH 1 el editor quedará preparado para editar la forma 1. Usted se puede mover por la pantalla utilizando las teclas para el cursor. Pulsando la barra espaciadora se llenará una caja vacía o se vaciará una llena. Después de haber diseñado una forma, se la define pulsando <ESC>. La instrucción SETSH 1 les conferirá a las tortugas 1, 2 y 3 (las tortugas en curso) la nueva forma.

ASK permite dirigir una instrucción a una tortuga determinada sin cambiar las en curso. Pruebe con:

```
ASK 1 [FD 20]
```

y verá que sólo se mueve la tortuga 1. Ahora digite FD 20 y se moverán las tres, porque las en curso aún siguen siendo los números 1, 2 y 3.

A las tortugas además de un encabezamiento y una posición se les puede dar una velocidad. SETSP 30 le confiere a la tortuga en curso una velocidad de 30 en su dirección actual. Las tortugas mantendrán sus velocidades hasta que éstas sean cambiadas. Se pueden crear procedimientos, o crear dibujos en la pantalla, sin que por ello se alteren las velocidades. Para detener a las tortugas, se les da una velocidad de 0. Entrando al editor también se detendrán, puesto que la entrada del editor siempre destruye la visualización de gráficos, ya que comparten la misma área de memoria.

En el Atari existen 128 tonalidades diferentes de color a elegir. Se puede establecer el color de fondo, el color del lápiz y el color de la tortuga. Por ejemplo, SETBG 92 establecerá el fondo en verde. SETPC 0 23 establecerá el color del lápiz en naranja. El 23 es el código para naranja y el 0 es el número de lápiz. En LOGO Atari, la tortuga puede elegir entres tres lápices para escribir, si bien en este capítulo sólo utilizaremos el lápiz 0 (el lápiz por defecto). SETC 7 pondrá en blanco a la tortuga en curso. En el manual de Atari hay una tabla de colores y sus correspondientes códigos.

Diablos

El aspecto más original del LOGO Atari es su utilización de *diablos*. El LOGO puede detectar 21 *colisiones* y *eventos especiales*. La mayoría de éstos son colisiones entre tortugas o entre tortugas y líneas. Un diablo le dice al LOGO lo que ha de hacer cuando se produce una de estas colisiones. Por ejemplo, la colisión número 0 es cuando la tortuga número 0 cruza una línea dibujada con el lápiz número 0. Para preparar un diablo utilizamos la instrucción WHEN (cuando). Pruebe con:

```
CS
TELL 0
PD
FD 50
PU
RT 90
FD 100
RT 90
FD 20
RT 90
```

con la tortuga de cara a él. Ahora prepare un diablo WHEN con la instrucción siguiente:

```
WHEN 0 [BK 50]
```

Inmediatamente no sucede nada, pero ahora el diablo WHEN está presente dentro del ordenador manteniéndose alerta ante un posible evento 0. Ahora pruebe SETSP 30. La tortuga parte hacia la línea, pero cuando llega a ella se activa el diablo WHEN y la tortuga es repelida. Ésta continúa yendo a una velocidad de 30, pero cada vez que llega a la línea el diablo WHEN la rechaza.

Este diablo WHEN permanecerá en operación hasta que sea eliminado digitando WHEN 0 [ ]. Todos los diablos desaparecerán al digitar CS, si se produce un mensaje de error o al emplear el editor.

Es una incomodidad tener que recordar todos los códigos para las distintas colisiones, de modo que hay dos primitivas para servirle de ayuda: OVER <númerodetortuga> <númerodelápiz> produce el número para la colisión entre esa tortuga y una línea dibujada con ese lápiz, y TOUCHING <númerodetortuga 1> <númerodetortuga 2> produce el número del diablo para una colisión entre esas dos tortugas.

A la caza de tortugas

He aquí un conjunto de procedimientos para encerrar una tortuga dentro de una caja. Cada vez que la tortuga toca el borde de la caja (WHEN OVER 0 0) un diablo llama al procedimiento GIRAR. Éste hace que la tortuga retroceda 10 unidades y luego efectúe un giro al azar. (RANDOM, junto con un número, N, produce un número al azar entre 0 y N−1 inclusive.)

```
TO ATRAPAR
  DIBUJAR.TRAMPA
  HOME
  WHEN OVER 0 0 [GIRAR]
  SETSP 50
END

TO DIBUJAR.TRAMPA
  CS
  PU
  SETPOS [-50 -50]
  PD
  CUADRADO
  PU
END

TO CUADRADO
  REPEAT 4 [FD 100 RT 90]
END

TO GIRAR
  BK 10
  RT RANDOM 45
END
```

Los diablos también se pueden utilizar para vigilar la palanca de mando. De los 21 eventos especiales que hemos mencionado, el evento 3 se produce cuando se pulsa el botón de la palanca, y el 15 cuando cambia la posición de la misma. La instrucción JOY 1 genera un número entre −1 y 7 que corresponde a la posición de la palanca (en la puerta 2). Defina JOYH de este modo:

```
TO JOYH
  IF(JOY 1)<0[STOP]
  ASK 0[SETH 45*JOY 1]
END
```

y ponga entonces en movimiento a la tortuga con SETSP 50; por último, prepare un diablo WHEN:

```
WHEN 15 [JOYH]
```

Ahora se pueden utilizar las palancas de mando para controlar el encabezamiento de la tortuga 0.

Se puede generar más de una instrucción WHEN al mismo tiempo, pero no estarán activas simultáneamente. Mientras un diablo está ocupado (es decir, cuando se produce su evento), los otros están inactivos. Por tanto, puede que se produzcan algunas colisiones sin que se las detecte.

La forma de tratar este problema consiste en hacer que cada diablo establezca las velocidades en 0 y ejecutar un procedimiento continuo que vigile que esto suceda. Adaptemos el programa anterior para aplicar esta técnica:

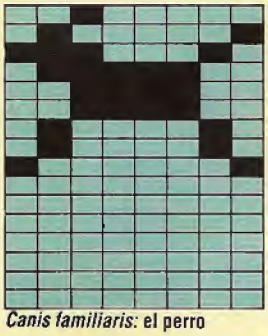
```
TO ATRAPAR
  DIBUJAR.TRAMPA
  HOME
  WHEN OVER 0 0 [SETSP 0]
  SETSP 50
  VIGILAR
END

TO VIGILAR
  IF :SPEED=0[VERIFICAR]
  VIGILAR
END
```

En este procedimiento, SPEED da el valor de la velocidad de la tortuga en curso. El procedimiento VERIFICAR debe determinar cuál es el evento que se ha producido, llevar a cabo las acciones necesarias y después restaurar las velocidades. En este caso sólo estamos interesados en un evento, pero ilustra la forma de programar este método.

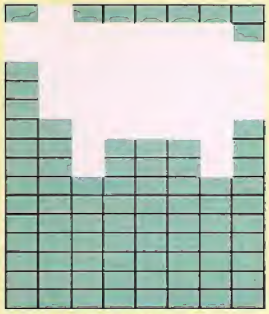
```
TO VERIFICAR
  IF COND OVER 0 0 THEN[TURN]
  SETSP 50
END
```

La instrucción COND y un número da una salida verdadera *(true)* si se ha producido un evento de ese número. COND sólo puede comprobar un evento en el momento en que el LOGO ejecuta la línea que lo contiene.

Kevin Jones



Canis familiaris: el perro



Liz Dixon

Ovis aries: la oveja

Empieza el juego

El juego en desarrollo

Ian McKinnell

¡Otra vez en el redil!

Agrupando ovejas

He aquí un juego que utiliza muchas de las características que hemos descrito. El jugador emplea la palanca de mando para controlar a un perro que está persiguiendo a dos ovejas por un campo. Si las ovejas tocan la cerca, retrocederán y girarán. Si chocan entre sí, girarán al azar. Si el perro toca a las ovejas, éstas girarán 90° hacia la derecha. Pulsando el botón de la palanca se dibuja una pequeña jaula en el rincón inferior izquierdo del campo. Volviendo a pulsar el botón la jaula se borra. La tarea del perro consiste en conducir a las ovejas hasta el interior de la jaula.

```
TO PERSEGUIR
  EST.VAR
  ASK :TORTUGA[EST.PANTALLA]
  EST.DIABLOS
  EMPEZAR
  VIGILAR
END

TO EST.VAR
  MAKE"CERCA 0
  MAKE"TORTUGA 0
  MAKE"OVEJA1 3
  MAKE"OVEJA2 2
  MAKE"PERRO 1
  MAKE"VERDE 92
  MAKE"NARANJA 23
  MAKE"NEGRO 0
  MAKE"BLANCO 7
END

TO EST.PANTALLA
  CS
  FS
  SETBG :VERDE
  HT
  PU
  SETPOS [-150 -80]
  PD
  SETPC 0 :MARRON
  RECT 160 300
  PU
END

TO RECT :LADO1 :LADO2
  REPEAT 2[FD :LADO1 RT 90 FD :LADO2 RT 90]
END

TO EST.DIABLOS
  WHEN OVER :OVEJA1 :CERCA[SETSP 0]
  WHEN OVER :OVEJA2 :CERCA[SETSP 0]
  WHEN TOUCHING :OVEJA1 :OVEJA2[SETSP 0]
  WHEN TOUCHING :PERRO :OVEJA1 [SETSP 0]
  WHEN TOUCHING :PERRO :OVEJA2 [SETSP 0]
  WHEN 3[SETSP 0]
  WHEN 15[JOYH]
END

TO JOYH
  IF (JOY 1)<0[STOP]
  ASK :PERRO[SETH 45*JOY 1]
END

TO EMPEZAR
  EST :OVEJA1 1[-150 20] 45 :BLANCO
  EST :OVEJA2 1 [150 20] 315 :BLANCO
  EST :PERRO 2[0 0]0 :NEGRO
  EST.VELOCIDADES
END

TO EST :NO :FORMA :POS :ENCABEZ :COLOR
  TELL :NO
  PU
  SETSH :FORMA
  SETC :COLOR
  ST
  SETPOS :POS
  SETH :ENCABEZ
END

TO EST.VELOCIDADES
  ASK :OVEJA1[SETSP 10]
  ASK :OVEJA2[SETSP 10]
  ASK :PERRO[SETSP 60]
END

TO VIGILAR
  IF SPEED=0[VERIFICAR]
  VIGILAR
END

TO VERIFICAR
  IF COND OVER :OVEJA1 :CERCA[ASK :OVEJA1
  [BK 20 RT 90]]
  IF COND OVER :OVEJA2 :CERCA[ASK :OVEJA2
  [BK 20 RT 90]]
  IF COND TOUCHING :OVEJA1 :OVEJA2[CHOCAR]
  IF COND TOUCHING :PERRO :OVEJA1[ASK
  :OVEJA1[RT 90]]
  IF COND TOUCHING :PERRO :OVEJA2[ASK
  :OVEJA2[RT 90]]
  IF COND 3[ASK :TORTUGA[DIBUJAR.JAULA]]
  EST.VELOCIDADES
END

TO CHOCAR
  ASK :OVEJA1[SETH RANDOM 360]
  ASK :OVEJA2[SETH RANDOM 360]
END

TO DIBUJAR.JAULA
  PU
  SETPOS[-150 -30]
  PX
  SETH 90
  REPEAT 2[FD 50 RT 90]
  PU
END
```

Ejercicios de LOGO

1. Modificar el juego de agrupar las ovejas de modo que el perro se controle utilizando el teclado en vez de la palanca de mando.
2. Escribir un programa para un juego en el cual usted está al mando de una nave espacial. Los meteoritos se precipitan hacia usted, y debe apartarse de su camino y sobrevivir el mayor tiempo posible. Le brindamos algunas pistas para ayudarlo. Utilice un sprite para la nave y los otros para los meteoritos. Emplee diablos WHEN para verificar las colisiones. Los meteoritos se mueven a una velocidad constante pero al azar. La nave se puede controlar mediante la palanca de mando.

Un digno sucesor

El Plus/4 supera a su antecesor, el Commodore 64, en varios aspectos: BASIC, memoria y programas incorporados

Sus fabricantes afirman que el Commodore Plus/4 se venderá al mismo tiempo que el Commodore 64, al cual no pretende sustituir. Pero el nuevo modelo ofrece tantas mejoras respecto al 64, que si consigue éxito en el mercado desplazará por completo a su antecesor.

El Plus/4 utiliza el microprocesador 7501, que es un desarrollo del 6502. Este chip está diseñado de tal modo que puede acceder a más de 64 K de memoria. Esto significa que la máquina tiene espacio para un BASIC muy satisfactorio, conservando al mismo tiempo su RAM libre para el usuario. Hay 64 K libres para utilizar con programas en BASIC, si bien esta cantidad se reduce a 50 cuando se emplean gráficos. Esto es más de lo que posee cualquier otro micro personal, a excepción del Sinclair QL (véase p. 981) y el Advance 86a (véase p. 829).

En el Plus/4 se ha implementado una versión excelente del BASIC Microsoft, y son particularmente dignas de mención las instrucciones para gráficos y sonido. En la modalidad de gráficos, la instrucción DRAW genera puntos o líneas y mediante la instrucción PAINT se puede rellenar con color cualquier forma dibujada. La instrucción BOX dibuja cuadrados y rectángulos sólo con líneas o con color sólido. La instrucción CIRCLE es particularmente versátil. Además de dibujar círculos, se pueden crear óvalos especificando la altura y la anchura del óvalo. Se pueden dibujar partes de óvalos para producir arcos, simplemente especificando en la instrucción posiciones de principio y final.

Todas las instrucciones operan normalmente en una pantalla con una resolución de 320×200 puntos. Esta resolución es la misma que en el Commodore 64, pero el Plus/4 realmente sobresale por su selección de colores. Puede mostrar 120 colores distintos en la pantalla al mismo tiempo, además del negro. Los mismos se crean a partir de 15 tonalidades básicas, cada una de las cuales se puede visualizar en ocho niveles diferentes de brillantez. Lamentablemente, el Plus/4 no puede producir sprites.

Las instrucciones para controlar el sonido son bastante estándares. La instrucción SOUND toca una nota de altura y duración especificadas. Una instrucción separada, VOL, especifica uno de ocho ajustes para el volumen de cada canal de sonido. Todo el sonido se emite a través del altavoz del televisor. El Plus/4 sólo proporciona dos canales de sonido, si bien el BASIC permite especificar tres. Este "tercer" canal es, en realidad, una facilidad de ruido y cualquier nota a la que se otorgue esa referencia de canal se reproducirá como tal. Esto es muy útil para los juegos, en los cuales se requieren efectos sonoros especiales.

Se han incluido instrucciones para mejorar el cuerpo principal del BASIC. Una instrucción AUTO producirá automáticamente números de línea en la introducción de programas; RENUMBER renumerará los números de línea de los programas, y VERIFY verificará que los programas se hayan salvado correctamente en cassette o en disco. Hay muchas instrucciones nuevas para trabajar con discos, y Commodore evidentemente desea venderles unidades de disco a un elevado número de usuarios de su nuevo ordenador.

Chris Stevens

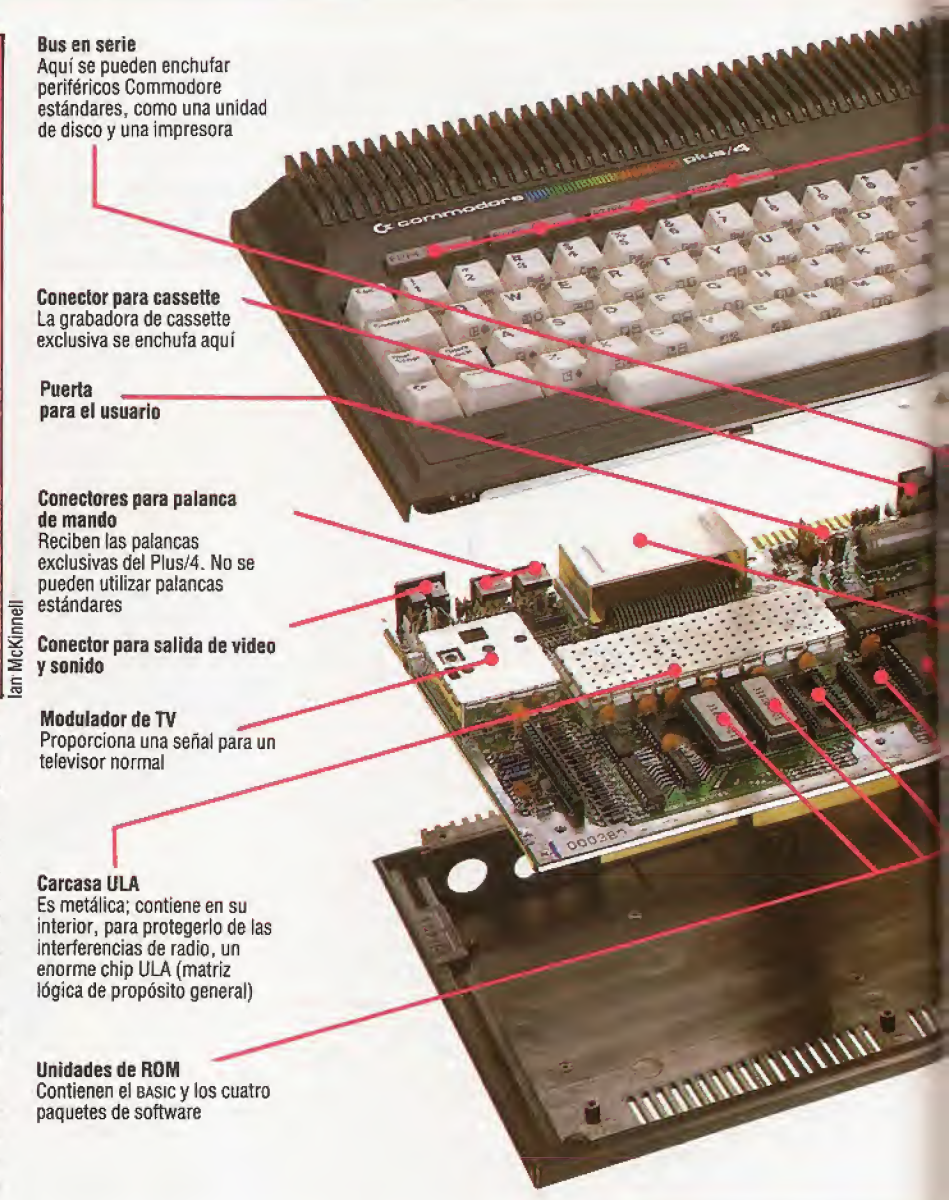
Apostando al futuro
El sucesor de Commodore, largo tiempo esperado, para el Commodore 64, posee todas las características que se están volviendo estándares en los micros más recientes: apariencia tipo MSX y racimo para cursor, memoria grande y software incorporado. Sin embargo, la competencia de ventas es intensa y los potenciales compradores están más informados que nunca. Commodore no está dando por segura su preeminencia en el mercado, tal como reflejan claramente el aspecto y las características del Plus/4

El Plus/4 tiene una visualización de textos de 40×25 caracteres. El usuario puede especificar dos puntos de la pantalla para que actúen como las esquinas de una "ventana". Todo el texto, como listados e instrucciones, aparecerá entonces dentro de esa ventana, sin alterar el resto de la pantalla.

Las teclas del Plus/4 son muy sensibles al tacto, requiriendo sólo una mínima presión para que sean registradas. A numerosos caracteres, tales como @, =, +, − y £ se les asignan teclas propias y se puede producir un juego completo de caracteres para gráficos desde el teclado. En la parte superior de éste hay cuatro teclas de función y cuando se enciende la máquina éstas quedan preparadas para producir las instrucciones utilizadas más comúnmente. Las teclas de función permiten que el usuario defina una instrucción nueva de hasta 128 caracteres para cada tecla. A partir de las cuatro teclas se pueden producir ocho funciones diferentes empleándolas con la tecla de cambio (Shift).

La generosa cantidad de espacio de memoria permite que el Plus/4 posea software de aplicaciones incorporado. Se suministran cuatro programas: un paquete de tratamiento de textos, hoja electrónica, base de datos y gráficos. Estos programas

QL contra Plus/4
Objetivamente, no cabe ninguna comparación: el QL, con sus microdrives incorporados, su mayor memoria, su SuperBASIC y su excelente software es, de acuerdo a los estándares del mercado, una buena adquisición; el Plus/4, con grabadora de cassette, se destaca sólo por la calidad de su teclado. Una apreciación objetiva se inclinaría sin vacilar por el QL; sin embargo, es posible que los usuarios adopten una decisión basada en la fidelidad a una marca de prestigio



Ian McKinnell

están diseñados para trabajar de forma conjunta.

Lamentablemente, el procesador de textos es bastante difícil de utilizar. El Plus/4 sólo puede visualizar 40 caracteres en una línea de su pantalla, pero muchas impresoras pueden imprimir 80 caracteres por línea. Para adaptarse a esta anchura, el contenido de la pantalla se desplaza al llegar a la columna 37 y sigue desplazándose hasta alcanzar la columna 77; entonces vuelve a saltar a la primera columna. El programa posee instrucciones de formato para establecer márgenes y justificar texto, pero sólo entran en vigor cuando se imprime el texto. Asimismo, posee instrucciones SEARCH y REPLACE para localizar frases o palabras determinadas dentro de un documento y reemplazarlas si fuera necesario. El máximo de texto que se puede entrar es de 99 líneas. Con los 77 caracteres de tamaño de línea esto representa un máximo de 1 500 palabras, lo cual no es suficiente para aplicaciones serias.

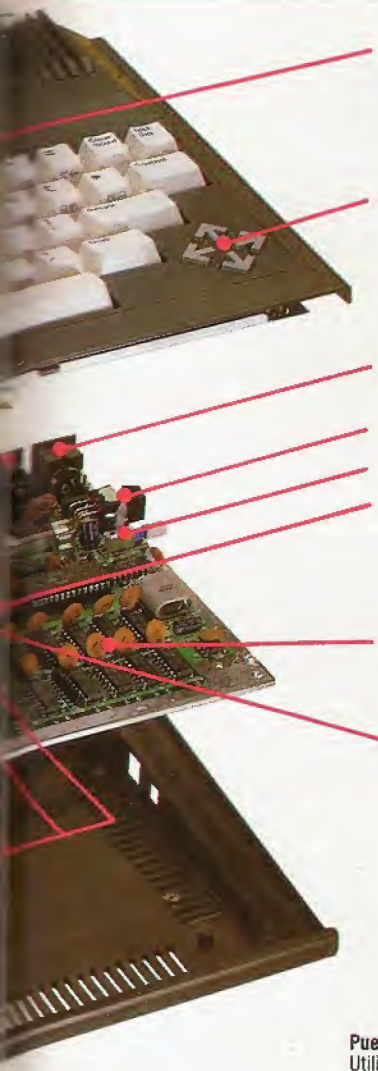
El programa de hoja electrónica es más fácil de utilizar que el de tratamiento de textos, si bien sufre, asimismo, de las limitaciones que supone una visualización en pantalla de 40 columnas. Ello significa que sólo puede mostrar tres celdas de la hoja electrónica a lo ancho de la pantalla y 12 a lo largo, aun cuando le es posible manejar modelos de hasta 17 celdas de ancho por 50 de largo.

El programa para gráficos es más bien decepcionante. Todo lo que hace es transformar un conjunto de cifras de la hoja electrónica en una especie de torpe gráfico de barras, formado a partir de gráficos de bloques, y transferirlo al procesador de textos. Mediante el mismo se lo puede visualizar o imprimir.

Tanto el procesador de textos como la hoja electrónica se pueden utilizar con la máquina estándar, pero la única forma de guardar sus resultados es disponiendo de una unidad de disco, lo cual significa que son de poca utilidad para las personas que dependan del almacenamiento en cassette. El último programa, el de base de datos, requiere una unidad de disco para su uso. Trabaja definiendo un formato estándar que se graba en disco como registros vacíos. Todos los datos se entran luego en los registros vacíos. Esto implica que un disco sólo se puede utilizar para una base de datos y el formato de la información no se puede modificar una vez que se han grabado datos. Cada base de datos puede contener hasta 999 registros, cada uno de ellos con un máximo de 17 campos de 38 caracteres. En líneas generales, el software es decepcionante: además de ser demasiado tosco para un uso de gestión serio, exige que el usuario personal adquiera una unidad de disco.

Muchos usuarios personales se sentirán más satisfechos con el monitor de código máquina incorporado, Tedmon, que con el software. El monitor es una gran ayuda para el programador de lenguaje máquina y se pone en funcionamiento mediante la instrucción MONITOR.

Commodore está produciendo numerosos accesorios para el Plus/4. Para muchos usuarios, el más importante de ellos será la grabadora de cassette. Al igual que otros micros Commodore, el Plus/4 requiere una grabadora fabricada especialmente por Commodore. Ésta emplea un enchufe diferente al de las más antiguas, por lo cual es necesario adquirir una grabadora nueva. El Plus/4 utiliza, asi-



Teclas de función
Se pueden programar con dos instrucciones de hasta 128 caracteres

Teclas del cursor

Conector de fuente de alimentación

Interruptor On/Off

Botón de Reset

CPU 7501

64 K de RAM

Puerta para ampliación
Utilizada por software en cartucho y por una unidad de disco "rápida"

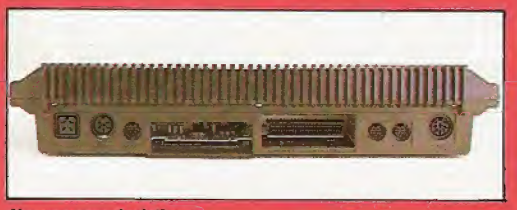
Chris Stevens

Teclas de función

Racimo del cursor

Botón de Reset (puesta a cero)

Puerta para ampliación/Puerta para el usuario



Nuevas características
A quienes ya poseen experiencia en máquinas Commodore les interesará saber que el Plus/4 posee tanto tecla Escape como Reset. Otros incentivos son el racimo de las teclas para el cursor, las teclas de función (programables desde BASIC) y Help. Los conectores para la fuente de alimentación, para cassette y para palanca de mando son diferentes de los del C64 y el Vic-20, al igual que las puertas para ampliación y para el usuario

COMMODORE PLUS/4

DIMENSIONES
67 x 203 x 338 mm

CPU
7501, 0,9 o 1,8 MHz

MEMORIA
64 K de RAM, 64 K de ROM

PANTALLA
Texto: 40 x 25. Gráficos: 320 x 200 en 121 colores

INTERFACES
2 conectores para palanca de mando, interface en serie, interface para cassette, puerta para cartucho/en paralelo

LENGUAJES DISPONIBLES
BASIC

TECLADO
Tipo máquina de escribir con 67 teclas (cuatro son de función)

DOCUMENTACION
Manuales bien concebidos que enseñan a programar y a utilizar el software incorporado

VENTAJAS
Versión de BASIC particularmente acertada, con buenas instrucciones para gráficos y facilidades de estructuración. 64 K completos de memoria

DESVENTAJAS
Requiere grabadora de cassette exclusiva y palancas de mando no estándares. El software incorporado es pobre

Un sobrecosto oculto

El software basado en ROM del Plus/4 comprende un procesador de textos, una hoja electrónica, una base de datos y una utilidad para gráficos. El principal punto fuerte de los paquetes es el hecho de estar incorporados; sin embargo, no se pueden emplear sin unidad de disco, lo que incrementa sustancialmente el costo de la máquina

Ian McKinnell

mismo, un conector diferente para palancas de mando.

Se está ofreciendo para la máquina una versión ligeramente mejorada de la lenta unidad de disco Commodore. La firma también está desarrollando una unidad de disco "rápida" que se enchufará en la puerta para cartuchos de la máquina, para proporcionar velocidades más cercanas a las de las unidades normales.

Son al menos cinco las impresoras que funcionarán con el micro: una de rueda margarita, dos matriciales comunes, una matricial en color y una impresora/plotter de cuatro lápices.

El Commodore Plus/4 se está vendiendo a un precio ligeramente más alto que el 64, pero su BASIC mejorado y su espacio de memoria extra hacen que represente una buena compra. En un futuro capítulo analizaremos con detalle cómo trabaja esta nueva versión del BASIC Commodore. La falta de software seguirá siendo un problema hasta que la máquina consiga establecerse en el mercado. Pero una empresa con las cifras de ventas de Commodore no debería tener ningún problema para conseguir un sólido apoyo de software por parte de las firmas especializadas más importantes.

Clones en acción

Ahora veremos cómo utilizar las características del "Vu-Calc" para efectuar cálculos de pagos de hipotecas o préstamos bancarios

El gran poder de los programas de hoja electrónica, incluso de un paquete tan sencillo como el *Vu-Calc* de Psion, es la forma en que permiten aplicar fórmulas complejas a los datos. Como veremos, con el *Vu-Calc* es posible construir algunos modelos interesantes y útiles, a pesar del hecho de que este programa en particular proporciona muy poco en cuanto a fórmulas incorporadas. En realidad, la única fórmula incorporada es su capacidad para "sumar" (es decir, sumar entre sí los contenidos de) bloques de celdas; ésta se indica colocando un signo @ delante de la dirección de la celda.

Las hojas electrónicas más perfeccionadas contienen fórmulas incorporadas muy sofisticadas, que el usuario puede llamar por su nombre. La ventaja de un sistema de este tipo es que realmente no hace falta conocer cómo trabajan estas fórmulas. Si se desea utilizar una fórmula de hipoteca con el *Multiplan*, por ejemplo, para calcular los pagos previstos por la compra de una vivienda en distintos períodos de pago (supongamos, 15, 20 y 25 años), simplemente se llama a la fórmula y se entran los datos correspondientes. El *Multiplan* calcula entonces todas las respuestas.

Con el *Vu-Calc* estos mismos cálculos pueden exigir mucho más tiempo y esfuerzo. El usuario debe construir por sí mismo las fórmulas necesarias y luego entrarlas en la máquina. El *Vu-Calc* también le impone al usuario una serie de limitaciones. Posee un máximo de 28 columnas, de modo que el modelo más grande que se puede construir, con cada columna representando un mes, cubrirá un período de poco más de dos años. La exactitud también puede representar un problema; el *Vu-Calc* trabaja sólo con valores enteros (de números enteros) e ignora las cifras que aparezcan después de una coma decimal, de modo que 99,9 se tomará como 99. No obstante, sí permite entrar valores y operaciones aritméticas en cualquier lugar del modelo. Por ejemplo, si el cursor está situado en una celda vacía (H5, p. ej.), se puede entrar 500*2 en la línea de instrucción. A continuación, si se pulsa la tecla Enter, se visualizará el resultado correspondiente (1 000) en la celda H5.

Otra característica negativa del *Vu-Calc* es el modo en que se editan las fórmulas. Un paquete "elegante", como el *Lotus 1-2-3*, utiliza para la edición una tecla de función. La pulsación de la tecla coloca automáticamente el contenido de la celda que contiene el cursor en la línea de instrucción. *Vu-Calc* posee una instrucción EDIT (#E) que se emplea para modificar una fórmula, pero la fórmula se debe volver a digitar cada vez que se utiliza la facilidad de edición. Si está trabajando con una fórmula larga y se da cuenta de que se ha olvidado entrar un paréntesis, no existe modo alguno de insertarlo: es inevitable volver a digitar toda la línea. Lo único que hace la instrucción EDIT es decirle al programa que borre la fórmula antigua de una celda e inserte luego la nueva.

Sin embargo, la utilización de la instrucción REPLICATE (#R) con una fórmula permite realizar un modelado bastante más complejo. Vamos a suponer que se desea ampliar el ejemplo del presupuesto doméstico (véase p. 1172) para anticipar los aumentos inflacionarios del presupuesto doméstico de alimentos, dando por sentada una tasa de inflación uniforme del 0,5 % mensual. Efectuar los cálculos necesarios con lápiz y papel sería una tarea que, a todas luces, consumiría mucho tiempo. Con el paquete *Vu-Calc* se puede realizar rápidamente mediante el empleo de una fórmula y REPLICATE.

Para llevar a cabo la operación deseada, debe indicarse al *Vu-Calc* que incremente su presupuesto mensual inicial (p. ej., 20 000 ptas) en un 0,5 por ciento. Las hojas electrónicas más sofisticadas hacen que esto resulte sencillo, mediante la utilización de una instrucción GROW BY (incrementar en un), pero el *Vu-Calc* requiere que el usuario entre las operaciones aritméticas que se deben llevar a cabo. Con el objeto de que el programa reconozca una fórmula que contiene direcciones de celdas, la fórmula debe ir precedida con un $ o un %. Éstos son dos símbolos elegidos de forma arbitraria que no tienen ninguna relación ni con dólares ni con porcentajes, sino que le dicen al programa que las direcciones de celdas son significativas en la fórmula que se está considerando, y estas direcciones pueden ser relativas (%) o bien absolutas ($). Una referencia a una celda absoluta le dice al *Vu-Calc* que busque y actúe sobre un valor de una celda, independientemente de la posición de ésta.

Para ver lo que hace una "dirección relativa", retomemos nuestro ejemplo. La fórmula para "incrementar" el presupuesto en un 0,5 por ciento es %B3*100,5/100, donde % indica una dirección de celda relativa y B3 es la dirección de la celda que contiene el valor que representa el presupuesto mensual para alimentos. Habiendo digitado esta fórmula en la celda B4, hemos entonces de copiar la fórmula para obtener el resultado para todo el año. B4 visualizará el resultado numérico de la fórmula; la fórmula propiamente dicha aparece en la parte inferior de la hoja de trabajo cuando el cursor está en B4. La instrucción REPLICATE #R,B4,B5:B14 nos da el resultado deseado (B4 contiene la fórmula, B5:B14 define la serie de celdas a través de las cuales se produce la reproducción). Los resultados se muestran casi al instante y nuestro modelo de hoja electrónica tendrá el siguiente aspecto:

	1	2	3	4	5
A			ENE	FEB	MAR
B	Pres. alim.		20 000	20 100	20 200

La visualización muestra por qué se utiliza la celda B3 para retener la cifra del presupuesto mensual inicial: la etiqueta se extiende a través de las columnas uno y dos, de modo que empezamos en la columna tres para crear una visualización pulcra. Observe que todas las cifras son valores enteros; la cifra de marzo debería ser 20 200,5 pero la hoja electrónica "redondea", de modo que la misma se visualiza exactamente como 20 200. La cifra de abril en realidad sería 20 301,502, pero *Vu-Calc* la tomará como 20 301. Dado que el aumento inflacionario se va haciendo mayor mes a mes, del mismo modo la discrepancia entre el valor real y la cifra visualizada se irá volviendo también mayor.

Este sencillo ejemplo demuestra el efecto de la instrucción REPLICATE cuando se utiliza con direcciones de celdas relativas. Cada vez que el programa escribe la fórmula en la siguiente celda por la derecha, la fórmula cambia de acuerdo con esto. Nuestra fórmula original en B4 era %B3*100,5/100. Esta fórmula se reproduce en B5 como %B4*100,5/100, en B6 como %B5*100,5/100, y así sucesivamente. En cada caso, el número de columna de la dirección de celda se incrementa en razón de uno. La reproducción a través de una columna hacia abajo produce el mismo efecto en las direcciones (es decir, E1 se convierte en F1, etc.). Si en vez de direcciones relativas hubiésemos utilizado direcciones absolutas ($), este "cambio" en las direcciones de celdas no se habría producido, sino que a través de todas las celdas se habría reproducido la misma fórmula y el valor en cada una de ellas sería idéntico al valor mostrado en B4.

Intentemos ahora emplear el mismo modelo para predecir los gastos mensuales realizados por una empresa en materias primas, empezando con 10 000 000 de ptas por mes e incrementando en un 0,5 % mensual durante dos años. ¿Cuánto más costarían las mercancías si se las comprara a mediados del segundo año? Utilizando nuestro modelo, esto se puede calcular muy rápidamente.

Cambie el valor en B3 por 10 000 000, desplazando el cursor hasta B3 y digitando la nueva cifra. Ahora utilice la instrucción REPLICATE para extender la fórmula desde B14 hasta B26, para conformar los 24 meses completos. Las hojas electrónicas más sofisticadas mostrarán los nuevos resultados en el momento en que se cambie el valor de B3. No obstante, con el *Vu-Calc* se deben volver a calcular los resultados (que por el momento aún se basan en la fórmula antigua) mediante el empleo de la instrucción CALCULATE, #C. El programa calcula entonces los nuevos valores y visualiza la respuesta que requerimos en la celda B20. Si prueba este ejemplo, verá que la cantidad es 10 993 100 ptas: un incremento de casi un millón de pesetas. Esta no es una cifra exacta, puesto que todos los números se redondean, pero es suficientemente aproximada como para darle una idea del efecto de la inflación durante este período.

A modo de último ejemplo del tipo de problemas de una sola fila, vamos a tomar una fórmula más complicada, diseñada para calcular la reducción de saldo de una deuda de tarjeta de crédito o préstamo bancario de 100 000 ptas, sobre la cual se está pagando un interés del 27 % anual. Suponiendo que usted esté devolviendo 8 000 pesetas al mes, ¿cuándo lo terminará de pagar? La información necesaria para calcular esto está en el principal del préstamo, más el interés para el mes, menos el pago mensual. De modo que si digitamos 100 000 en B1, la fórmula será %B1+%B1*,27/12-80. Reproduzca la fórmula a través de las 28 columnas del modelo, explore la fila para hallar el punto en el cual la cantidad se vuelve positiva, y habrá hallado el punto en el cual el saldo estará pagado por completo y usted estará con saldo a su favor en caso de continuar los pagos mensuales. De acuerdo a nuestro modelo, esto ocuparía 16 meses. A modo de complemento, también posee una pulcra visualización de su saldo pendiente mes a mes, suponiendo que mantenga constantes sus pagos de 8 000 ptas.

Relativamente absolutas

En este simple modelo hemos utilizado la instrucción REPLICATE a través de las filas C, D y E; para claridad del ejemplo, reproducimos la fórmula en cada celda. La celda C3 se ha reproducido de forma absoluta a todo lo largo de la fila, de modo que en todas las celdas C aparece la misma fórmula, A5/12, con el mismo resultado: 45 000 ptas. Las fórmulas de las celdas D4 y E3, sin embargo, se han reproducido de forma relativa, por lo cual todas las referencias a celdas de las fórmulas cambian de una celda a otra a lo largo de la fila, con resultados consiguientemente variables

	1	2	3	4	5
A			**INGRESOS ANUALES=540 000**		
B			**ENERO**	**FEBRERO**	**MARZO**
C	INGRESOS DOMÉSTICOS		A5/12 45 000	A5/12 45 000	A5/12 45 000
D	GASTOS DOMÉSTICOS		DATOS 40 000	D3*1,01 40 400	D4*1,01 40 800
E	SALDO MENSUAL		C3-D3 5 000	C4-D4 4 600	C5-D5 4 200

Liz Dixon

Control de energía

Esta vez construiremos un convertidor de digital a analógico para añadir a nuestro sistema para la puerta para el usuario

Para este proyecto hemos optado por utilizar un convertidor de digital a analógico ya hecho en un chip, si bien se puede construir un circuito a partir de distintos componentes. La salida analógica de este chip, el DAC, se deposita con un amplificador en un segundo chip. La salida de éste se conduce directamente a una salida y a través de un condensador y un control de nivel a la otra salida.

Paso uno: Corte la carcasa para acomodar las dos conexiones del bus del sistema. También se puede cortar un conector de empalme para utilizarlo en futuros proyectos.

Paso dos: Corte la veroboard (de 30 agujeros por 16 franjas). Ahora haga los cortes de pistas tal como indica el diagrama. Primero suelde en su lugar los conectores de chip, luego los cables de enlace. A continuación se deben colocar en su lugar los dos condensadores. No tiene importancia de qué lado se instalen. Si desea instalar el conector para ampliación del bus, entonces suéldelo en su sitio ahora e instale el cable plano.

Paso tres: Coloque los cuatro conectores en la carcasa, con el potenciómetro. Haga las conexiones entre éstos con el alambre estañado. Lleve los tres cables aéreos hasta la placa de circuitos.

Paso cuatro: Enchufe los dos chips y el convertidor estará ya completo. Observe que los dos chips no se enchufan con la misma orientación: el chip convertidor D/A se debe colocar de modo que la muesca quede a la izquierda cuando se lo mira desde arriba, con el conector macho del bus en la parte superior; la muesca del chip amplificador debe ir a la derecha.

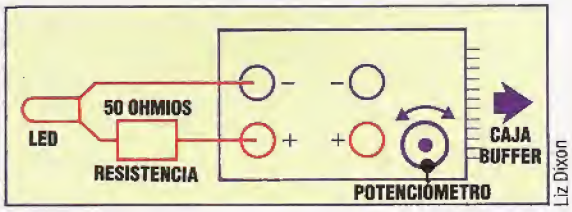
Después de haber construido el convertidor de digital a analógico y haber verificado cuidadosamente todas las conexiones, puede probar la unidad. El convertidor D/A convertirá en un voltaje cualquier valor binario de ocho bits que se coloque en el registro de datos de la puerta para el usuario. Este voltaje sale de la unidad de dos formas. En el par de conectores de salida de CD (corriente directa) se obtiene un voltaje de CD comprendido entre 0 y +2,5 V, correspondiente a los valores digitales de la puerta para el usuario de 0 a 255. El otro par de conectores de salida es para permitirnos simular una salida de CA (corriente alterna). El nivel de voltaje global se controla mediante un potenciómetro y se puede ajustar para adaptarlo a la entrada requerida por otro aparato.

Para probar la unidad, podemos desarrollar un experimento sencillo para alterar la brillantez de un LED. Para hacerlo se requieren estos pasos:

Paso uno: Conectar en serie un LED, del tipo utilizado en la caja buffer original (véase p. 1003), a una resistencia de 50 ohmios.

Paso dos: Conectar la unidad convertidora D/A directamente a la puerta para el usuario y proporcionarle energía de la forma habitual.

Paso tres: Conectar el LED y el circuito de la resistencia a través de los conectores de salida CD de la caja D/A y ejecutar este programa:




```
 10 REM**  PROGRAMA DE PRUEBA DE D/A PARA CBM 64  **
 20 RDD=56579:REGDAT=56577
 30 VL=127
 40 POKE RDD,255:REM TODAS SALIDA
 50 POKE REGDAT,VL
 60 PRINT VL
 70 GET A$
 80 IF A$<>"Z" AND A$<>"X" THEN 70
 90 IF A$="X" THEN DV=1
100 IF A$="Z" THEN DV=-1
110 VL=VL+DV
120 IF VL<256 AND VL>=0 THEN 50
```

```
 10 REM**** PROGRAMA DE PRUEBA DE D/A PARA BBC ****
 20 RDD=&FE62:REGDAT=&FE60
 25 valor=127
 30 ?RDD=255:REM TODAS SALIDA
 40 REPEAT
 55 ?REGDAT=valor
 57 PRINTvalor
 60 A$=GET$
 62 IF A$<>"Z" AND A$<>"X" THEN 60
 65 IF A$="X" THEN dv=1 ELSE dv=-1
 67 valor=valor+dv
 70 UNTIL (valor>255 OR valor<0)
```

Este sencillo programa dedica a salida las ocho líneas de la puerta para el usuario al colocar 255 (es decir, 11111111) en el registro de dirección de datos. En el registro de datos de la puerta para el usuario se coloca entonces un valor inicial de 127. Pulsando las teclas Z o X, el valor del registro de datos disminuye o bien aumenta respectivamente. El programa termina cuando el valor del registro cae fuera de la escala comprendida entre 0 y 255.

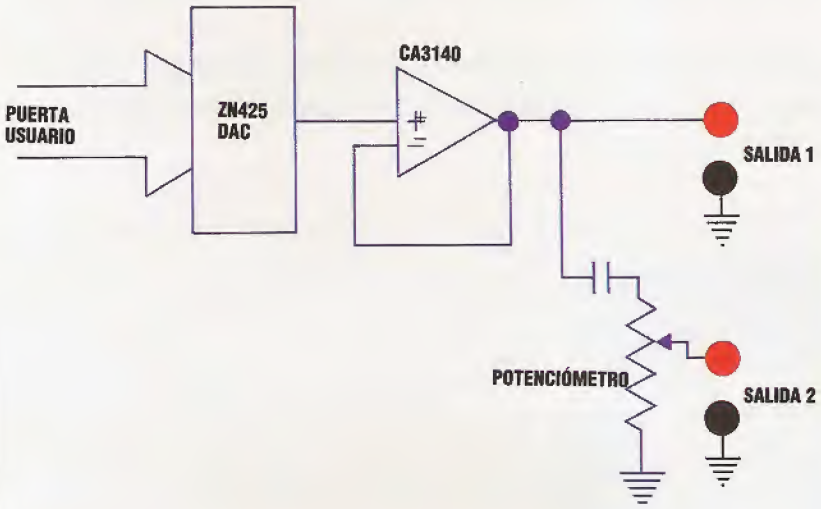
Incrementando el valor *digital* presente en el registro de datos podemos producir voltajes *analógicos* crecientes para el LED. A medida que el voltaje vaya aumentando hasta un nivel aceptable, el LED comenzará a brillar, primero tenuemente y luego con más intensidad a medida que se vaya aumentando el voltaje, hasta alcanzar una brillantez máxima cuando el valor presente en el registro de datos de la puerta para el usuario sea de 255.

Si su LED no se ilumina, intente invertir las conexiones a la caja convertidora D/A, antes de verificar la existencia de cualquier otro fallo. A diferencia de una bombilla normal, que se ilumina independientemente de en qué dirección fluya la corriente, un LED sólo se encenderá cuando ésta fluya en una determinada dirección.

Extraer el dígito
El convertidor de digital a analógico que vemos aquí incorpora un mando de potenciómetro excesivamente grande. Esto no es necesario, y debería arreglarse como para poder usar una perilla de control. Los lectores ya sabrán que cuando uno sale a comprar componentes electrónicos, con frecuencia tiene que quedarse con lo que consigue y adaptarlo a sus necesidades

Ian McKinnell

Circuito completo

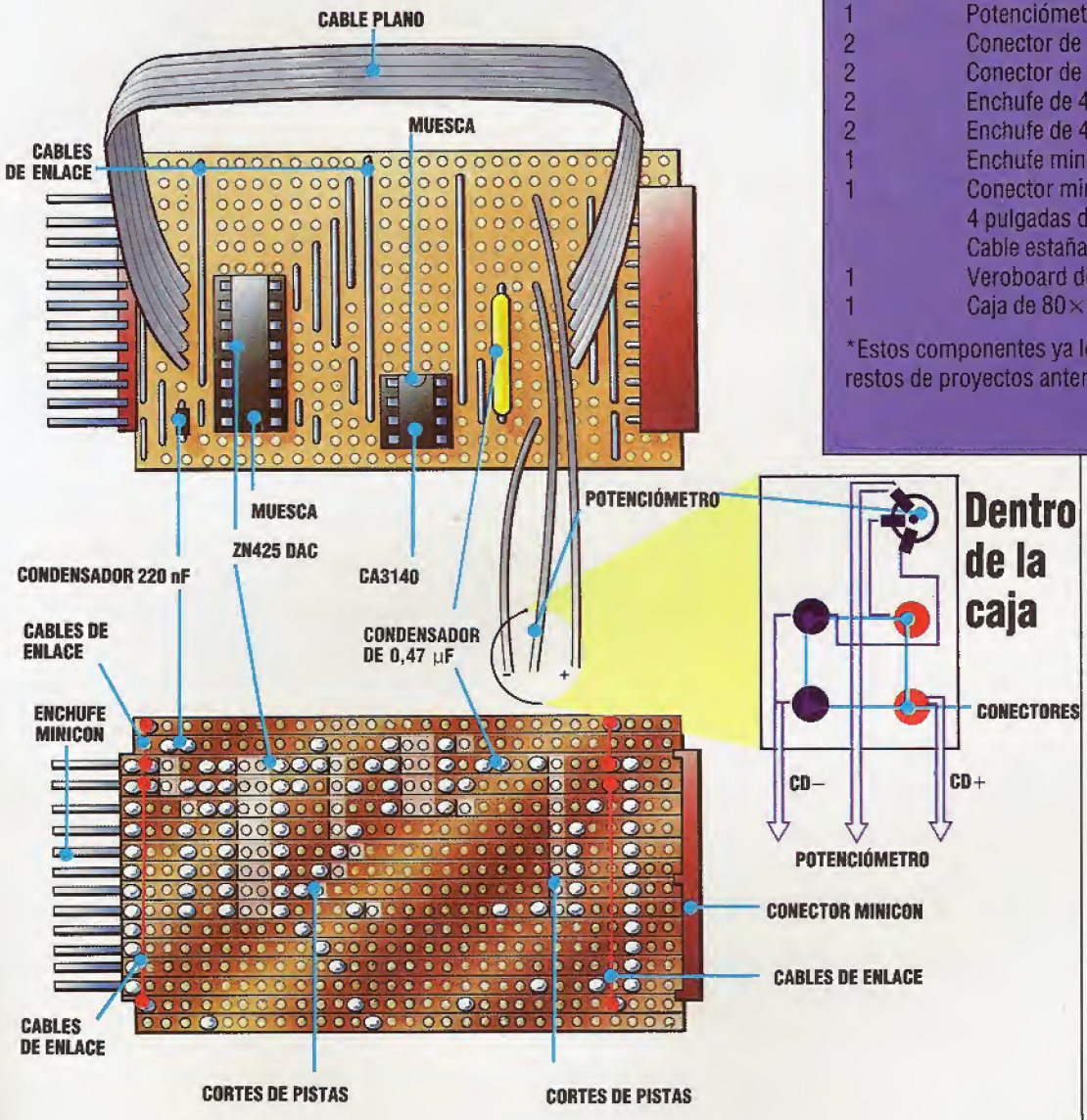


Debe ser muy cuidadoso respecto a los detalles de trazado, en especial a la ubicación y limpieza de los cortes de pistas. Compruebe con regularidad su trabajo mediante el tester, inserte los componentes pasivos y los cables de enlace primero, utilice la soldadura discretamente y el estaño de soldar, y preste especial atención al posicionamiento de los chips

Lista de componentes

Cantidad	Artículo
1	ZN425 DAC
1	Amplificador CA3140
1	Conector DIL de 16 patillas
1	Conector DIL de 8 patillas
1	Condensador de 220 nF
1	Condensador de 0,47 µF
1	Potenciómetro rotativo de 10 K
2	Conector de 4 mm rojo
2	Conector de 4 mm negro
2	Enchufe de 4 mm rojo
2	Enchufe de 4 mm negro
1	Enchufe minicon de 12 vías
1	Conector minicon de 12 vías
	4 pulgadas de cable plano de 5 vías*
	Cable estañado pelado*
1	Veroboard de 50 agujeros×24 franjas
1	Caja de 80×61×41 mm

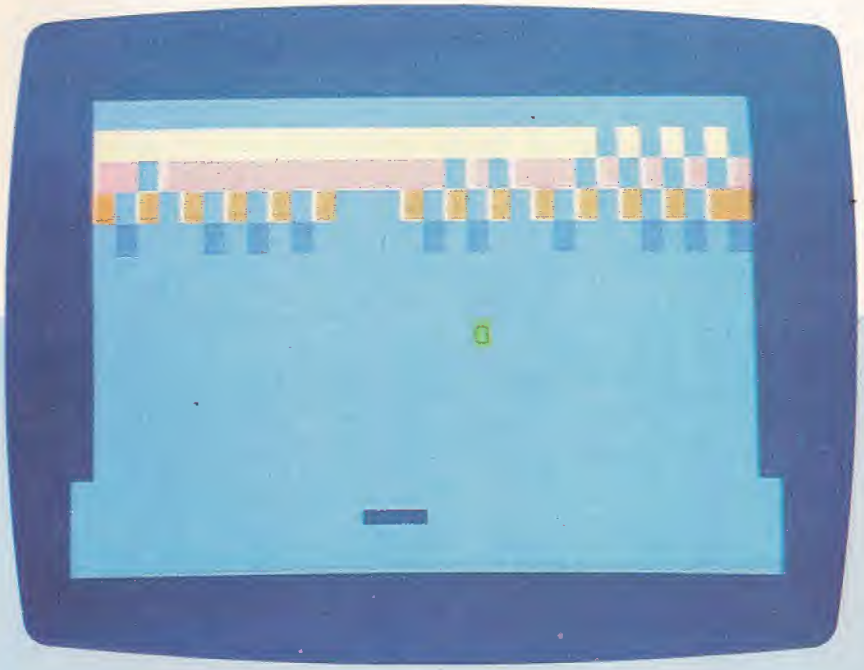
*Estos componentes ya los debe de tener como restos de proyectos anteriores.



Kevin Jones

La pared

En esta ocasión presentamos el popular juego de "romper la pared", tan habitual en las máquinas recreativas comerciales. He aquí la versión para el Dragon



El objetivo del juego es muy sencillo: intentar destruir una pared de ladrillos con ayuda de una pelota que debe ser relanzada con su raqueta. Cada ladrillo que rompa le proporciona un punto. Cuando el muro ha sido destruido por completo, aparece otro nuevo. El usuario dispone de diez pelotas para intentar conseguir el mayor número posible de puntos. Utilice la palanca de mando o las teclas A, S y la barra espaciadora para desplazar la raqueta.

```
 10 REM ********************************
 20 REM *            LA PARED          *
 30 REM ********************************
 40 S=0:NB=0
 50 GOSUB 530
 60 V=V+DV:H=H+DH
 70 POKE B+1024,CN
 80 B=V*32+H
 90 L=PEEK (B+1024)
100 IF L<>223 AND L<>128 THEN DV
    =-DV:K=0:S=S+1
110 POKE B+1024,CB
120 IF V=13 AND ABS(R-29-B)>1THEN
    320
130 IF V=13 AND H>2 AND H<29 THEN
    POKE B+1024,CN:H=H+CH
140 IF V=13 OR V=1 THEN DV=-DV
150 IF H=1 OR H=30 THEN DH=-DH
160 ON JS GOSUB 220,250
170 IF R<446 THEN R=446
180 IF R>475 THEN R=475
190 PRINT@ R,R$;
200 IF S/120=INT (S/120)AND K=0
    THEN GOSUB 670
210 GOTO 60
220 L=INT (JOYSTK(0)/2+445)
230 IF ABS(L-R)>1 THEN R=R+2*SGN
    (L-R):CH=SGN(L-R)
240 RETURN
250 D$=INKEY$
260 D=2*((D$="A")-(D$="S"))
270 IF D$=" " THEN D0=0
280 IF D<>0 THEN D0=D
290 R=R+D0
300 CH=SGN(D0)
310 RETURN
320 NB=NB+1
330 FOR I=1 TO 5
340 SOUND 1,1
350 FOR J=1 TO 50
360 NEXT J,I
370 IF NB=11 THEN 410
380 POKE B+1024,CN
390 GOSUB 760
400 GOTO 60
410 IF S>R1 THEN R1=S
420 PRINT@ 166,"PUNTOS :";S;
430 PRINT@ 230, "RECORD :";R1;
440 PRINT@ 294, "OTRA ?";
450 FOR I=1 TO 100
460 D$=INKEY$
470 NEXT I
480 D$=INKEY$
490 IF D$="" THEN 480
500 IF D$<>"N" THEN 40
510 CLS
520 END
530 CLS
540 PRINT@ 203, "PAL.MANDO ?";
550 D$=INKEY$
560 IF D$="" THEN 550
570 IF D$="S" THEN JS=1 ELSE JS=2
580 CLS 6
590 R=461
600 FOR I=1024 TO 1055
610 POKE I,128
620 NEXT I
630 FOR I=1 TO 12
640 POKE I*32+1024,128
650 POKE I*32+1055,128
660 NEXT I
670 FOR I=1057 TO 1086
680 POKE I+32,159
690 POKE I+64,239
700 POKE I+96,255
710 POKE I+128,175
720 NEXT I
730 K=1
740 R$=CHR$(223)+CHR$(223)+CHR$
    (211)+CHR$(211)+CHR$(211)+CHR$
    (223)+CHR$(223)
750 CB=79:CN=223
760 V=13:DV=-1
770 H=RND(28)+1
780 B=V*32+H
790 DH=(RND(2)-1.5)*2
800 B1=B:D0=0
810 RETURN
```

Tránsito libre

En este capítulo del curso sobre el assembly del 6809 trataremos de las técnicas generales de programación en lenguaje máquina

Si un programa es escrito en código *reubicable* o *independiente de la posición*, éste puede alojarse en cualquier lugar de la memoria y ejecutarse sin más cambios. Lo cual es importante en los sistemas multiusuario y multitarea, en los que varios programas pueden estar ocupando la memoria al mismo tiempo, y para que el uso de la memoria sea eficiente, el sistema operativo deberá poder cargarlos en los lugares más convenientes. Hasta en los sistemas monousuario más sencillos suele tener su importancia la posibilidad de mantener bibliotecas de subrutinas y de construir un programa con módulos autosuficientes, en cuyo caso las rutinas pueden cargarse no siempre en el mismo sitio.

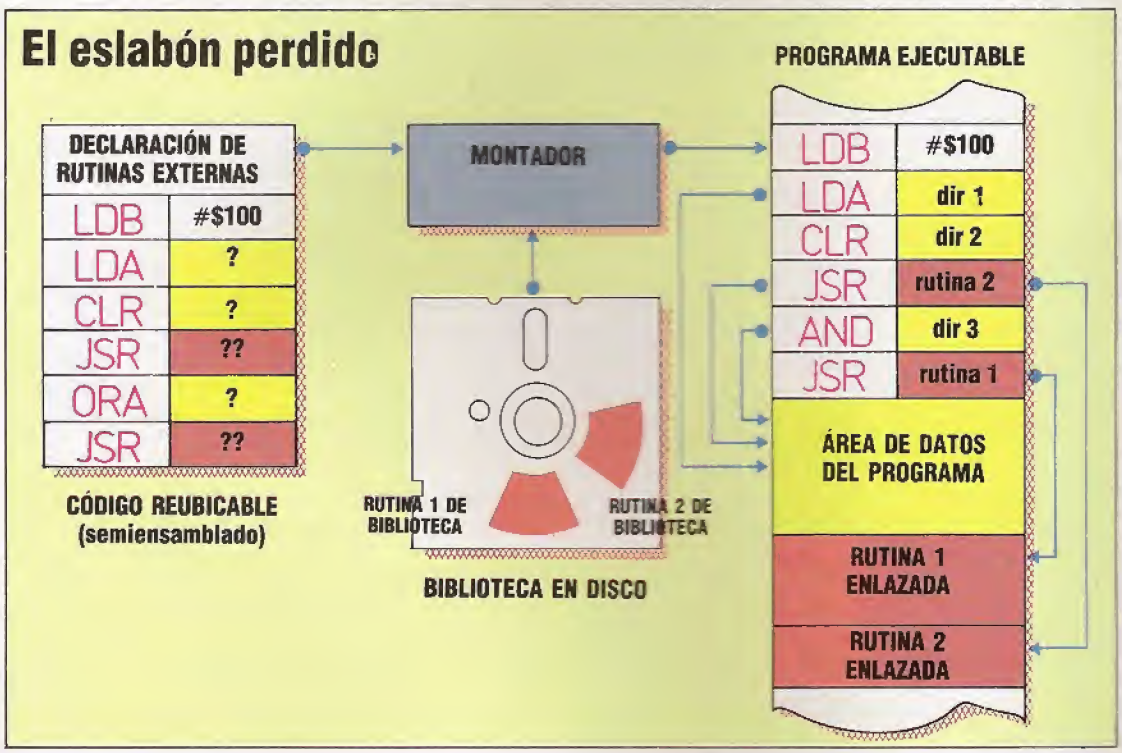
Muchos procesadores solventan la cuestión mediante un programa conocido como *montador (linking loader)*. El ensamblador elabora un código reubicable que prescinde de todas las referencias a direcciones absolutas de memoria; el montador pone en los lugares adecuados del programa las direcciones reales según va cargando el programa en la memoria. Y puesto que el montador mismo se ocupa de las direcciones, se asegura por completo la correcta transferencia de control entre los módulos. De este modo es posible escribir fragmentos de código en lenguajes diferentes que se compilen o ensamblen en el mismo código reubicable; así, por ejemplo, programas en PASCAL podrían llamar rutinas pertenecientes a una biblioteca en FORTRAN. El 6809 admite también esta solución, del todo necesaria cuando la construcción de programas es por módulos. El 6809 además facilita el proceso, pues permite la escritura en código totalmente reubicable de manera directa, por lo que no se necesita la fase adicional de insertar direcciones.

Un punto esencial en la escritura de código reubicable consiste en referenciar todas las direcciones mediante un *desplazamiento* (offset) relativo al contador del programa (PC). Un programa puede emplear una dirección de dos maneras: o como dato o como destino en una transferencia de control. Las instrucciones de bifurcación (BRA, BSR, etc.) calculan sus destinos en base a desplazamientos relativos al PC y son necesarias en toda transferencia de control dentro del programa del usuario. Las instrucciones de transferencia absoluta (JMP y JSR) sólo se emplearán para destinos que ocupan siempre el mismo lugar en la memoria, como ocurre con las rutinas del sistema operativo.

La tarea más difícil está en independizar todas las referencias a posiciones de datos. El 6809 lo consigue permitiendo la indexación mediante el PC. La instrucción:

```
LDA     OFFSET,PC
```

sumará ese offset o desplazamiento con signo al valor actual del PC para obtener la *dirección efectiva*. El problema reside en calcular el desplazamiento correctamente, para lo que es necesario hallar la diferencia entre la dirección de los datos y el valor



Liz Dixon

Montador
En los grandes sistemas, los programas en código máquina se cargan en la memoria mediante el montador (*linking loader*). Esta utilidad del sistema operativo recoge el código semiensamblado (o sea, sin direcciones absolutas) del ensamblador, y determina cuál es la mejor dirección libre para el dato o estado actual del sistema. Emplea esta dirección para sustituir las direcciones simbólicas que el ensamblador dejó en el programa sin dirección absoluta, para enlazar a continuación la rutina de biblioteca que el programador desea incorporar al programa; esta rutina la cargará en cargada procedente de la biblioteca en disco y agregará al programa. Por último, el montador pasa el programa, para su ejecución, al sistema operativo.

actual del PC, sin olvidar que el PC queda incrementado tan pronto como la instrucción a la que apunta es cargada en el procesador para su ejecución. Cuando una instrucción está siendo ejecutada, el PC queda, como se sabe, apuntando a la instrucción siguiente.

El procedimiento se complica a causa de la variación en las longitudes de las instrucciones en el 6809, que van de un byte hasta cinco. Por ejemplo:

```
LDX OFFSET,Y
```

emplea un byte para el opcode y otro byte para el *post-byte*, o sea, el byte empleado en cualquier instrucción indexada para especificar el registro índice a usar, independientemente de que se tenga en cuenta o no la indirección. El desplazamiento emplea dos, uno o ningún byte, según su tamaño. Los desplazamientos "cero" y los que pueden expresarse en cinco bits pueden ser incorporados en el post-byte (aunque esta opción no es muy bien manejada por algunos ensambladores). Desplazamientos más grandes necesitan un byte más (si pueden ser expresados en ocho bits) o dos bytes más. Cuando el PC es usado con indexación, los desplazamientos especiales de cinco bits o ninguno no son admisibles. La instrucción:

```
LDY OFFSET,X
```

necesita incluso un byte más, dado que el opcode de LDY es de dos bytes.

Si le agrada escribir en lenguaje assembly y está familiarizado con las decisiones de qué direcciones de datos usar y los lugares donde colocar las subrutinas, verá muy pronto que las tareas de manejar los opcodes, convertir las direcciones en formato de dos bytes y calcular desplazamientos de salto, le resultarán completamente naturales. Más simple que esta escritura en assembly manual es la adquisición de un ensamblador para que la realice él, porque en todo caso habrá de calcular la longitud de cada instrucción. Muchos ensambladores emplean la sigla especial PCR (Referido al Contador del Programa) para que el ensamblador haga uso del PC como un registro índice y calcule el desplazamiento. Por ejemplo:

```
DATITM    FCB  0
.....
          LDA  DATITM,PCR
```

Simulación de terminales

Damos en esta lección una subrutina que emplea esta técnica para la emulación de diversos terminales, de tal modo que el programa escrito para un determinado terminal pueda ser ejecutado también en otro sistema. Las diferencias entre terminales se hacen más evidentes en la codificación empleada para controlar las diversas funciones de pantalla, como son el borrado de pantalla y el posicionamiento del cursor. Los códigos pueden ser de control (caracteres cuyo ASCII es inferior a 32) o secuencias de escape, consistentes en el carácter de Escape (27 en ASCII) seguido de cualquier otro carácter o secuencia de caracteres. Nuestra sencilla rutina sólo nos permitirá la sustitución de un carácter de control por otro, o de un único carácter a continuación de un Escape por otro. Pero esta rutina es muy útil para mostrar cómo funciona una emulación. Hay dos tablas: una contiene los caracteres de control y la otra los caracteres de Escape. Si un programa genera un carácter de control, pongamos el caso, este carácter sirve de desplazamiento de la tabla de donde se sacará el carácter que ha de visualizarse.

Dado que la rutina es totalmente reubicable, la podemos añadir a cualquier programa y en cualquier posición. Suponemos que hay una rutina de sistema operativo (OUTCH) que envía el carácter contenido en el acumulador A a la pantalla, y empleamos JMP para acceder a esta rutina que ha de hallarse en una posición fija de la memoria. Se observará que falta por dar la directiva ORG, aunque no tenga efecto alguno. El carácter a visualizar debe encontrarse en A.

Longitud de las instrucciones

El problema de calcular la longitud de las instrucciones no se limita al empleo de un direccionamiento con PCR. A menudo es necesario conocer la longitud total de una rutina que debe ajustarse a un espacio limitado de la memoria, por ejemplo, en una ROM. Cualquier libro sobre el assembly del 6809, o cualquier manual del ensamblador, debe incluir una tabla de mnemotécnicos junto con sus datos asociados. Por cada expresión mnemotécnica, el dato incluirá el opcode correspondiente, la longitud total de la instrucción (aunque a veces esto no es posible, en cuyo caso la longitud mínima será seguida del signo +), el número de ciclos de reloj que emplea la instrucción al ser ejecutada, y el efecto de la instrucción sobre los flags del código de condición.

Veamos aquí las reglas generales para hallar la longitud de una instrucción, para así poder escribir en código compacto:

1) La mayoría de opcodes son de un solo byte; los que afectan directamente al contenido de S e Y (salvo en LEA) y algunos que afectan a U (como LDY y STS) son de dos bytes.
2) Todo direccionamiento indexado requiere un post-byte, y a veces uno o dos bytes más, según el tamaño del desplazamiento.
3) Los datos que van a continuación de un opcode en el modo inmediato son de uno o dos bytes, según el tamaño del registro empleado.
4) Las direcciones han de ser de un byte cuando se trata de la página directa (posiciones del $00 al $FF, generalmente) y de dos bytes en los restantes casos. No todos los ensambladores emplean apropiadamente el direccionamiento directo, causa por la cual una dirección requiere dos bytes cuando sería de esperar sólo uno.

Igualmente complejo es el problema del cálculo del tiempo empleado en la ejecución de cada instrucción, por la sencilla razón de que el tiempo depende a su vez de los bytes cuestionados, es decir, de la longitud de la instrucción. Pero es importante en aplicaciones en tiempo real y en el tratamiento de ciertos periféricos. El tiempo de cada instrucción se mide en ciclos de reloj (o al menos en el mínimo número de ciclos de reloj) que emplea la instrucción. Lo cual significa que el tiempo real em-

pleado depende también de la frecuencia de oscilación del reloj. La frecuencia de reloj más común para sistemas con 6809 es de 1 MHz por segundo (un millón de ciclos por segundo). O sea, cada ciclo tarda una millonésima de segundo. La instrucción inmediata:

```
LDA DATITM
```

que utiliza una dirección de 16 bits, emplea cinco ciclos, es decir, se ejecuta en cinco millonésimas de segundo. La instrucción:

```
PSHS PC,B,CC
```

emplea cinco ciclos, más otro ciclo por cada byte que pone en la pila. En este caso, el total es de nueve ciclos (recuerde que el contador del programa es de dos bytes).

Si un sistema no lleva incorporado un reloj en tiempo real, la única manera de medir el tiempo transcurrido será mediante una rutina software de demora. Tal rutina pone en funcionamiento una serie de instrucciones cuyos tiempos respectivos son conocidos de antemano, de tal modo que su suma proporciona el intervalo pedido. Tales intervalos suelen medirse en milisegundos (milésimas de segundo), por tanto no es necesario ser excesivamente precisos, pues un error de millonésima de segundo no es significativo. Suponiendo la velocidad a 1 MHz por segundo, la rutina de demora (Software Delay) que proporcionamos en esta página dará intervalos entre 1 y 255 milisegundos: el número exacto de milisegundos (ms) se coloca como un parámetro en A. La notación (A) significa el contenido del acumulador A.

Podemos expresar el cálculo de la constante COUNT como sigue:

Instrucción	Núm. de ciclos reloj	Número de veces ejecutadas	Tiempo empleado (ciclos de reloj)
PSHS B,CC	7	1	7
LDB #COUNT	2	(A)	(A)*2
DECB	2	(A)*COUNT	(A)*COUNT*2
BNE LOOP2	3	(A)*COUNT	(A)*COUNT*3
DECA	2	(A)	(A)*2
BNE LOOP1	3	(A)	(A)*3
PULS PC,B,CC	9	1	9

Lo que da un total de (A)*(7+5*COUNT)+16 ciclos de reloj. Para facilitar los cálculos, despreciaremos el 16. A un MHz por segundo, hay mil ciclos de reloj en un milisegundo, luego el tiempo total será (A)*1000 ciclos de reloj.

$$(A)*(7+5*COUNT) = (A)*1000$$
$$(7+5*COUNT) = 1000$$
$$5*COUNT = 973$$
$$COUNT = 195 \text{ (redondeado)}$$

Es factible hallar intervalos más precisos y usar registros de 16 bits si es necesaria una mayor escala, pero el principio de decrementar un registro un número determinado de veces será el mismo.

Rutina de simulación de terminal

```
ESCAPE  EQU   27
SPACE   EQU   32        (32: Espacio en ASCII)
OUTCH   EQU             Poner aquí la dirección del
                        sistema operativo
        ORG   $1000
CTABLE  RMB   32        Tabla de caracteres de control
ETABLE  RMB   128       Tabla de caracteres de Escape
EFLAG   FCB   0         Flag que indica si el último
                        carácter fue un Escape
DISPCH  PSHS  X         Guarda X
        TST   EFLAG,PCR Comprueba si el último carácter
                        fue un Escape
        BEQ   DISP1     En caso negativo, va a DISP1
        LEAX  ETABLE,PCR En caso afirmativo, carga la
                        dirección de ETABLE en X
        LDA   A,X       Toma el carácter de sustitución
                        por medio del carácter original
                        contenido en A considerado
                        como desplazamiento
        CLR   EFLAG,PCR Restablece EFLAG
        BRA   FINISH
DISP1   CMPA  SPACE     Carácter de control?
        BGE   FINISH    No lo es : va a FINISH
        CMPA  ESCAPE    Sí lo es : comprueba si es
                        Escape
        BEQ   ESCCH     Caso de serlo, va a ESCCH
        LEAX  CTABLE,PCR Carga la dir. de CTABLE en X
        LDA   A,X       Toma el carácter de sustitución
                        considerando como desplaz. el
                        car. cont. en A
        BRA   FINISH
ESCCH   INC   EFLAG,PCR Activa EFLAG para avisar que se
                        trata de un carácter Escape
FINISH  PULS  X         Restaura X
        JMP   OUTCH     Visualiza el car. cont. en A
        END             Nótese que al final de OUTCH, la
                        instrucción RTS devolverá el
                        control desde aquí al programa
                        que llamó a esta rutina
```

Rutina de demora software

```
                        Subrutina para demorar (A)
                        milisegundos
COUNT   EQU   195       Ver el cálculo
        ORG   $1000
DEMORA  PSHS  B,CC      Guarda los otros dos registros
                        afectados
LOOP1   LDB   #COUNT    Cuenta 1 ms
LOOP 2  DECB            No decrementando...
        BNE   LOOP2     ...hasta que B llegue a cero
        DECA            Decrementa A a cada ms...
        BNE   LOOP2     ...hasta que A alcance el cero
        PULS  PC,B,CC   Regresa
```

Decisión suprema

Los juegos de mayor aceptación suelen combinar elementos de juegos recreativos, de estrategia y de aventuras. "Psytron" es un buen ejemplo

Psytron es un juego complejo y absorbente que despliega unos excelentes gráficos de acción rápida, y cuyo dominio le llevará al jugador mucho tiempo. Éste asume el papel de Psytron, un dispositivo mitad hombre, mitad ordenador que dirige una colonia espacial en el planeta Betula 5. La colonia se compone de varios edificios, cada uno de los cuales posee una función específica. Los sistemas de apoyo a la vida son esenciales para los colonos humanos, y se requieren plantas energéticas para mantener el ordenador en funcionamiento. La instalación más importante es la planta energética principal, sin la cual toda la colonia quedaría paralizada. Mediante la utilización del teclado o bien de una palanca de mando, al jugador se le ofrece una "exploración" completa de 360° de la instalación. Cada edificio está dibujado cuidadosamente para conferirle mayor realismo a la panorámica.

Durante los primeros niveles de juego, *Psytron* se comporta de forma muy similar a cualquier otro juego recreativo. El usuario dispone de numerosas armas con las cuales repeler los ataques de invasores extraterrestres, y es en el cuarto nivel donde se hacen evidentes los elementos de estrategia del juego.

En el primer nivel, el Psytron controla un *droid*, que se emplea para destruir a los saboteadores de tres piernas que son "teletransportados" a la base en un intento de hacer volar las esclusas de aire que conectan entre sí los edificios de la colonia. El segundo y el tercer nivel de juego le ofrecen al jugador la oportunidad de derribar los platillos volantes de los extraterrestres; éstos se pueden coger uno por uno; no obstante, si se utiliza Disruptor, es posible eliminar de una sola vez todos los alienígenas que haya a la vista. Pero el Disruptor es algo inestable y hay un 10 % de probabilidades de que estalle cuando se haga uso de él.

Una vez llegado el cuarto nivel, el jugador tiene la oportunidad de tomar decisiones estratégicas, basadas en la cuantía de los daños que hayan sufrido las instalaciones de la colonia. A lo largo de este nivel las naves extraterrestres continuarán atacando, bombardeando estratégicamente zonas vitales de la base y lanzando saboteadores en misiones suicidas. No obstante, en este nivel se introduce el "tiempo congelado". Mediante la simple pulsación de la tecla Return, el jugador puede "congelar" la acción para poder recibir y procesar informes acerca de los daños. Los factores a considerar incluyen el número de miembros de la colonia que han resultado muertos o heridos, el nivel de los suministros disponibles y los daños infligidos a la planta energética. Estando en tiempo congelado se pueden realizar reparaciones y se pueden destinar miembros de la colonia a aquellas zonas en las que sean más efectivos. En esta etapa es necesaria una cuidadosa administración de los recursos: Psytron debe tener en cuenta el hecho de que los grupos de reparación consumirán más alimento y más oxígeno que los colonos que no trabajen, y que podría ser necesario abandonar algunos edificios con el objeto de economizar combustible, aire y provisiones.

El quinto nivel le presenta al jugador la oportunidad de comunicarse con una nave de suministros, que podría hacerle llegar a la colonia provisiones vitales. Por consiguiente, se ha de tener especial cuidado en asegurarse de que los sistemas de acoplamiento no hayan resultado dañados en los ataques enemigos.

En el sexto y último nivel los factores estratégicos revisten máxima importancia. El único objetivo en esta etapa consiste en sobrevivir durante una hora, conservando la base intacta mediante la utilización de todas las facilidades que han sido introducidas en los niveles anteriores. Aumenta el número de naves atacantes, la acción se acelera y enseguida resulta evidente, a la luz de la sobrecogedora potencia de fuego de los atacantes, que es imposible mantener intactas todas las instalaciones de la colonia. Se deben tomar decisiones relativas a qué edificios se deben sacrificar; es de vital importancia, en particular, proteger la bahía de acoplamiento, dado que la misma permitirá el reabastecimiento de provisiones.

Los juegos por ordenador han recorrido un largo camino desde los días de *Space invaders*, y *Psytron* representa una alternativa exigente y absorbente a los juegos recreativos que hasta hace muy poco han venido dominando el mercado.

A través de los ojos del "droid" Estas escenas de *Psytron* muestran parte de la acción que se desarrolla en el primer nivel del juego. Mientras el *droid* persigue al saboteador a través de los corredores, la panorámica va girando a través de la base. Mediante la ventana situada en el rincón inferior derecho de la imagen, el jugador puede ver a través de los ojos del *droid*. Cuando el saboteador aparece en la pantalla, el jugador lo puede destruir accionando el pulsador de disparo

A la caza del saboteador — Saboteador localizado

Cortesía de Your Spectrum

Psytron: Para Spectrum de 48 K y Commodore 64
Editado por: Beyond Software, Competition House, Farndon Road, Market Harborough, Leics LE16 9NR, Gran Bretaña
Autores: Tayo Olowu y Paul Voysey
Palanca de mando: Opcional
Formato: Cassette (Spectrum), cassette o disco (Commodore 64)


mi COMPUTER